

T H E S E

présentée

A LA FACULTE DES SCIENCES
DE L' UNIVERSITE DE PARIS

Pour obtenir le titre de Docteur-Ingénieur

par

Samir ZRAICK

MACRO GENERATEUR RECURSIF POUR LE LANGAGE TAS-86

1ère partie : LE LANGAGE DU MACRO GENERATEUR

2ème partie : LOGIQUE INTERNE DU SYSTEME

Soutenue le 16.11.1970

devant la Commission d'Examen

MM. BLAQUIERE	) Examinateurs
LE CORRE	
DEBRAINE	

Président

T H E S E

présentée

A LA FACULTE DES SCIENCES
DE L' UNIVERSITE DE PARIS

Pour obtenir le titre de Docteur-Ingénieur

par

Samir ZRAICK

MACRO GENERATEUR RECURSIF POUR LE LANGAGE TAS-86

1ère partie : LE LANGAGE DU MACRO GENERATEUR

2ème partie : LOGIQUE INTERNE DU SYSTEME

Soutenue le 16.11.1970

devant la Commission d'Examen

MM. BLAQUIERE	)	Président
LE CORRE	}	Examineurs
DEBRAINE	}	



Avant de présenter ce travail, je tiens à remercier vivement M. BLAQUIERE d'avoir bien voulu accepter de présider le jury.

Je tiens à exprimer ma profonde reconnaissance à Monsieur P. DEBRAINE qui a dirigé mes travaux et qui, à tout moment, a su me prodiguer ses conseils et orienter mes recherches.

Je tiens aussi à remercier M. P. THELLIER, Directeur Général des Sociétés ECA-Automation et EUROSYSTEM, de m'avoir permis le développement de cette thèse dans le cadre de ces entreprises.

Je tiens à remercier M.F.GEDDA qui a été le premier utilisateur du macro-générateur et aussi le principal responsable du développement du langage MACRO CORAL.

Je remercie également Mme COHEN et le personnel du service de documentation de Saclay, grâce à qui j'ai pu présenter cette thèse à la date prévue.

TABLE DES MATIERES

	page
PREMIERE PARTIE - DESCRIPTION GENERALE	
INTRODUCTION GENERALE	1
DESCRIPTION SOMMAIRE DU LANGAGE TAS-86	4
 DEUXIEME PARTIE - LE LANGAGE DU MACRO-GENERA TEUR	
DESCRIPTION DU LANGAGE	1
FACILITES ADDITIONNELLES FOURNIES PAR LA MACRO GENERATEUR	9
APPELS DE MACROS	31
APPELS EMBOITES	39
 ANNEXE 1	AI-1
ANNEXE 2	AII-1
ANNEXE 3	AIII-1

PREMIERE PARTIE

"DESCRIPTION GENERALE"

CHAPITRE I

1 - INTRODUCTION GENERALE

1-1 - Définition

Un MACRO-GENERATEUR est un traducteur, capable d'interpréter et de traduire un programme écrit dans un MACRO-LANGAGE.

1-2 - Macro-instructions

Lorsqu'un programme est écrit en langage assembleur, le programmeur est censé spécifier chaque opération élémentaire, exécutable ou non, par une instruction symbolique ; celle-ci est traduite en forme binaire par l'assembleur. L'usage des macros dans le langage assembleur permet au programmeur d'associer à une séquence d'instructions symboliques un nom et d'utiliser postérieurement ce nom pour préciser la séquence d'instructions en question.

Prenons le cas où l'on spécifie les instructions pour effectuer $A = B + C$ en TAS-86 (Langage Assembleur pour la machine Telefunken TR-86).

B	B	Charger B
A	C	Ajouter C
C	A	Mémoriser en A

Au lieu d'écrire toutes les opérations élémentaires, nous pourrions définir une macro-instruction qui préciserait les opérations à effectuer

*ADD A, B, C

Un astérisque, précédant immédiatement le nom, indique une macro-instruction.

Nous pouvons ainsi définir une macro comme une instruction non-élémentaire. Une des tâches du macro-générateur est de remplacer une macro-instruction par ses opérations élémentaires.

Tous ces concepts seront décrits en détail, de façon bien plus précise, dans les chapitres suivants.

1-3 - Macro-langage

Un ensemble de macros peut être défini pour exécuter des fonctions standard, souvent requises, telles que :

- Gestion de bits et de caractères
- Définitions de tableaux et de structures
- Méthode d'accès à des éléments d'un tableau ou d'une structure
- Opérations arithmétiques
- Instructions du genre "if", "for", "do"
- Opérations d'entrée-sortie

Cet ensemble de macros constitue ce que nous appelons une bibliothèque de Macros-Système, accessible à tout programmeur. Les facilités fournies par l'usage de ces macros constituent les bases du Macro-Langage. Dans ce sens, il peut être considéré comme un langage évolué.

Cependant, la syntaxe d'un macro-langage est plus primaire car chaque opération est décrite par un appel de macro qui est moins compréhensible qu'une instruction dans un langage évolué.

Dans l'exemple très simple d'une addition, nous avons pour les deux cas :

Macro-langage

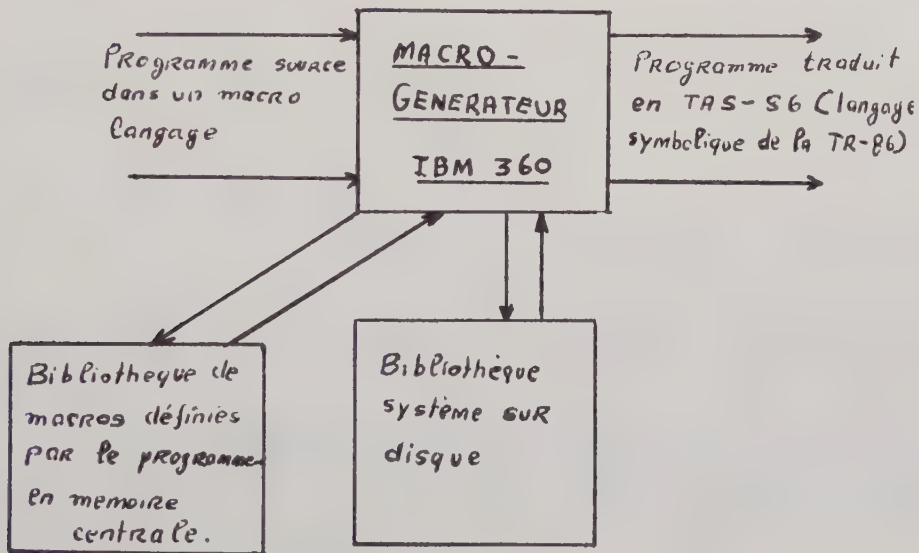
*ADD A, B, C

Langage évolué

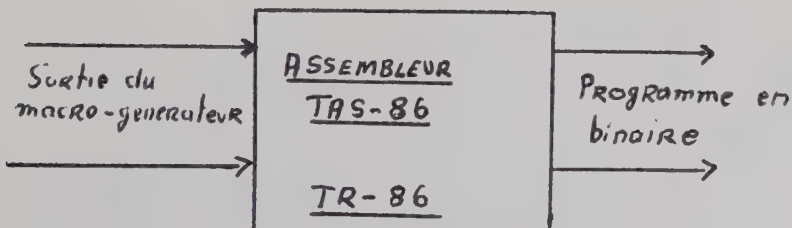
A = B + C

raison pour laquelle un macro-langage peut être considéré comme un état intermédiaire entre le pur symbolique et un langage évolué.

a) - MACRO-LANGAGE —→ TAS-86



b) TAS-86 —→ binaire



A part les macros système qui se trouvent dans la bibliothèque standard, chaque programmeur peut définir ses propres macros qui ne seront utilisés que par son programme.

2 - DESCRIPTION SOMMAIRE DU LANGAGE TAS-86

2-1 - Le fonctionnement du macro-générateur est entièrement indépendant de l'assembleur TAS-86. Dans les paragraphes suivants, nous donnerons un bref aperçu du langage TAS-86, pour permettre au lecteur de mieux suivre les exemples qui se trouvent dans le texte.

2-2 - Introduction

L'assembleur TAS-86 est extrêmement simple, le répertoire d'instruction de la machine TR-86 n'ayant que 18 codes opérations.

Le format général des instructions TAS-86 est :

étiquette = code opération opérande

Le symbole "=" précise l'existence d'une étiquette, qui est un identificateur formé de 1 à 6 caractères alphabétiques ou numériques, dont le premier est alphabétique.

Le code opération est représenté par le correspondant mnémonique de l'opération machine en question. Ces opérations seront décrites dans les paragraphes suivants .

L'opérande contient, dans la plupart des cas, une adresse qui détermine la variable affectée par l'instruction. Le format est libre, à condition que l'instruction soit perforée entre les colonnes 1 et 68 d'une carte.

2-3 - Descriptions des opérations machine

La machine TR-86 ne possède pas de registre d'index. Chaque mot contient 24 bits dont le premier est le signe.

Les deux registres opérationnels sont l'accumulateur A et le registre auxiliaire Q, de 24 bits chacun.

Les nombres négatifs sont représentés par le complément à 1. Dans les instructions, les 6 premiers bits précisent le code opération. Le septième détermine un adressage indirect s'il est égal à 1.

Le restant du mot représente une adresse sauf dans le cas des instructions de décalage.

2-31 - Chargement

B X

Cette instruction place dans l'accumulateur le contenu de l'adresse X

$\langle A \rangle := \langle X \rangle$

2-32 - Stockage

C X

Cette instruction stocke dans l'adresse X le contenu de l'accumulateur

$\langle X \rangle := \langle A \rangle$

2-33 - Stockage de la partie adresse

CA X

La partie adresse de l'accumulateur (bits 7 à 24) est stockée dans la partie adresse de X

$\langle X \rangle_{7-24} := \langle A \rangle_{7-24}$

2-34 - Intersection

ET X

Le résultat de l'intersection logique du contenu de l'accumulateur et de l'adresse X, effectuée bit par bit, est placé dans l'accumulateur

$$\langle A \rangle := \langle A \rangle \wedge \langle X \rangle$$

2-35 - Ou exclusif

AT X

Entièrement analogue à la précédente. Au lieu d'une intersection, un ou exclusif est effectué.

$$\langle A \rangle := \langle A \rangle \oplus \langle X \rangle$$

2-36 - Addition de la partie adresse

AA X

La valeur numérique X qui se trouve dans la partie adresse de l'instruction est ajoutée au contenu de l'accumulateur

$$\langle A \rangle := \langle A \rangle + X$$

Ex. AA 5U

La lettre U à la fin du nombre indique à l'assembleur que la valeur 5 est immédiate.

Si nous écrivons AA 5, nous obtiendrons dans la partie opérande une adresse qui contient la valeur numérique 5. A chaque fois qu'un nombre se termine par U, sa valeur reste immuable dans la partie adresse de l'instruction.

2-37 - Soustraction de la partie adresse

SA X

La valeur numérique X est retranchée du contenu de l'accumulateur

$$\langle A \rangle := \langle A \rangle - X$$

Ex. SA 5U

2-38 - Addition

A X

Le contenu de l'adresse X est ajouté au contenu de l'accumulateur

$$\langle A \rangle := \langle A \rangle + \langle X \rangle$$

2-39 - Soustraction

SB X

Le contenu de l'adresse X est retranché du contenu de l'accumulateur

$$\langle A \rangle := \langle A \rangle - \langle X \rangle$$

2-40 - Multiplication

ML X

Le résultat du produit du contenu de l'accumulateur et de l'adresse X est placé dans l'accumulateur et dans les 23 premiers bits de Q

$$\langle A . Q \rangle := \langle A \rangle . \langle X \rangle$$

2-41 - Branchement

S X

Un branchement est fait à l'adresse X

$$\langle F \rangle := X$$

F représente le compteur d'adresses.

2-42 - Branchement si zéro

SI X

Un branchement est fait à l'adresse X si l'accumulateur est égal à zéro. Autrement, le programme continue en séquence.

$\langle F \rangle := X$ si $\langle A \rangle = 0$

sinon

$\langle F \rangle$ reste immuable

2-43 - Branchement si négatif

SK X

Un branchement est effectué à l'adresse X, si l'accumulateur est négatif. Autrement, le programme suit en séquence.

$\langle F \rangle := X$ si $\langle A \rangle < 0$

sinon

$\langle F \rangle$ reste immuable

2-44 - Branchement à un sous-programme

SU X

Un branchement est effectué à l'adresse $X + 1$ et le contenu du compteur d'adresses est stocké en X

$\langle X \rangle 7 - 24 := \langle F \rangle$

$\langle F \rangle := X + 1$

2-45 - Décalage

L'instruction de décalage SH modifie les contenus de l'accumulateur et éventuellement du registre Q, selon les options précisées dans la partie adresse de l'instruction

Ex. SH L 15

Cette instruction effectue un décalage de quinze positions à gauche dans l'accumulateur.

Les options sont :

- L - précise un décalage à gauche. Si rien n'est précisé, un simple décalage à droite de l'accumulateur est adopté.
- K - précise un décalage cyclique, c'est à dire que les bits qui quittent les positions extrêmes des registres, sont reintroduits à l'extrême opposé.
- Z - précise que A et Q sont décalés ensemble, comme s'il s'agissait d'un seul registre de 48 positions.

Toutes ces options peuvent être combinées

Ex.

SH ZL 23

Cette instruction détermine un décalage dans A et Q de 23 positions à gauche. Les opérations restantes se réfèrent aux indicateurs lumineux de la machine. Nous ne les décrirons pas, car elles ne sont pas utilisées dans ce texte.

2-4 - Instructions pour l'assembleur

2-41 - Littéraux

L'assembleur TAS-86 remplace automatiquement un littéral par une adresse en mémoire dont le contenu est égal à la valeur numérique du littéral.

Les littéraux définis par leur représentation octale ou hexa-décimale se trouvent entre 2 apostrophes.

Ainsi les 3 instructions de chargement

```
B    12
B    '14'0
B    'C'H
```

entraînent la génération de la constante 12 en mémoire.

Une autre possibilité est de définir un littéral par sa représentation alpha-numérique, entre 4 apostrophes

Ex. B "12"

Cette instruction définit un littéral égal à la représentation interne des caractères 1 et 2, qui n'a aucun rapport avec la valeur numérique 12.

2-42 - Réservation de mémoire

En TAS-86, il n'existe pas d'obligation de déclarer explicitement des variables qui occupent un seul mot en mémoire. L'assembleur réserve automatiquement une adresse en mémoire pour la variable en question.

Si l'on veut déclarer un tableau, le nombre de mots à réserver apparaît entre parenthèses.

Ex. T A B L E = (20)

Vingt mots sont réservés dont l'adresse de tête est associée à TABLE.

2-43 - Définition de constante

Une constante est définie simplement en mettant sa valeur après l'étiquette.

Ex. V A R = 10
 V A R = 'A'H
 V A R = '12'0

L'exemple précédant montre 3 façons de définir une constante numérique.

2-44 - Commentaires

Un commentaire, seul, sur un carte, doit être précédé de deux signes moins "--". Si nous voulons l'insérer après la partie opérande, une virgule doit précéder les signes moins ",--".

Ex. -- commentaire sur carte --
 B TABLE , -- charger l'accumulateur --

2-45 - Instructions de début et fin de programme

La première instruction d'un programme a toujours la forme NM nom ; "nom" est un identificateur de 6 caractères et sert à nommer le programme. L'instruction qui indique à l'assembleur la fin du programme source est simplement ND,

Ex. NM PROG10 ,--nom du programme --
 Instructions TAS-86
 ND,

2-46 - Référence au compteur d'adresses

Lorsque nous désirons nous référer à une adresse relative à l'adresse de l'instruction exécutée, nous devons préciser une valeur numérique suivie du caractère "L" à la fin.

Ex. B 2L
 AA 1U
 S ADDR

L'instruction B 2L charge dans l'accumulateur le contenu
de * +2, c'est à dire, S ADDR

2-47 - Adressage indirect

Le caractère "E" suivant immédiatement le code opération précise
un adressage indirect. Il se traduit par la mise à 1 du bit 7, dans la con-
figuration binaire de l'instruction traduite.

Ex. BE ADRI

Dans ce cas, le contenu de ADRI détermine l'adresse de l'opérande.
Si le bit 7 de ADRI est égal à 1, le procès d'adressage indirecte se poursuit
d'une façon enchaînée, jusqu'à ce que l'on trouve le bit 7 égal à zéro.

DEUXIEME PARTIE

"LE LANGAGE DU MACRO-GENEREUR"

CHAPITRE II

DESCRIPTION DU LANGAGE

1 - DESCRIPTION

1-1 - Introduction

Etant une instruction non-élémentaire, une macro doit être définie comme une séquence d'instructions élémentaires.

Les macros incluses dans la bibliothèque système ne sont pas définies par le programmeur ; celui-ci, cependant, doit savoir comment les appeler.

Le macro-générateur offre à chaque programmeur la possibilité de définir ses propres macros.

Lorsqu'un programmeur définit une macro, pour son usage privé, le choix du nom de cette macro reste entièrement libre. Si le nom coïncide avec une autre définition, déjà existante dans la bibliothèque système, le nom du programmeur remplacera temporairement la définition standard. Cette procédure offre la possibilité d'adapter le langage à chaque cas particulier, de la façon la plus efficace.

Les règles de définition seront décrites dans les paragraphes suivants.

L'unité syntaxique qui décrit le format de la macro-instruction correspondante est désignée par le nom de mot prototype.

Sa définition formelle est :

$$\begin{aligned} \langle \text{mot prototype} \rangle &::= \{ \text{\$} \langle \text{nom} \rangle \langle b \rangle = \}^1_0 \langle b \rangle \langle \text{nom de macro} \rangle \\ \langle b \rangle &\{ \langle \text{liste de paramètres} \rangle \}^1_0 \\ \langle \text{nom de macro} \rangle &::= \langle \text{nom} \rangle \\ \langle \text{liste de paramètres} \rangle &::= \text{\$} \langle \text{nom} \rangle \langle b \rangle \{ \langle b \rangle \text{\$} \langle \text{nom} \rangle \langle b \rangle \}^{14}_0 \\ \langle \text{nom} \rangle &::= \langle \text{lettre} \rangle \{ \langle \text{lettre} \rangle | \langle \text{chiffre} \rangle \}^5_0 \\ \langle \text{lettre} \rangle &::= A | B | \dots | Z \\ \langle \text{chiffre} \rangle &::= 0 | 1 | \dots | 9 \\ \langle b \rangle &::= \text{\$} \langle b \rangle | \text{caractère vide} \end{aligned}$$

$\text{\$}$ représente un blanc.

Les **nombre**s indiqués pour les unités entre $\{ \}$ précisent les facteurs de répétition minimum et maximum pour l'unité correspondante, le **nombre** inférieur représentant le minimum. Il est évident que les unités qui admettent un facteur de répétition de zéro minimum sont **facultatives**. La fin de carte détermine la fin de l'instruction.

Exemples de mots prototypes :

```

$LABEL = ADD $PAR1, $PAR2, $PAR3
$CALL  = MOVE $SOURCE, $TAB, $L1, $L2
        END
        DECLTA $TAB, $D1M1, $D1M2
$SYMB  = BEGIN
    
```

Les règles de format pour le mot prototype sont très souples et, comme il a été précisé dans sa définition formelle, les blancs entre les différents champs sont toujours ignorés.

Dans le but d'une présentation claire nous recommandons que l'étiquette commence à la colonne 1, le nom de la macro ou l'opération à la colonne 10 et la liste des paramètres à la colonne 16.

Conformément aux règles TAS-86, un nombre maximum de 6 caractères, le premier étant alphabétique, est admis pour un identificateur, ce qui justifie la définition de <nom>. Dans le cas où la longueur maximum est excédée, seuls les 6 premiers caractères seront pris en compte.

Les paramètres doivent être séparés par des virgules et les blancs intermédiaires sont toujours ignorés.

1-3-2 - Restrictions à l'utilisation

Le nombre maximum de paramètres qui peuvent être définis dans une macro est de 15, comme il a été indiqué dans la définition de <liste de parametres> dans l'item précédent. Cette quantité est largement suffisante pour toutes les applications pratiques.

Le macro-générateur lit de la colonne 1 à la colonne 68. Cependant, si le programmeur le souhaite, des cartes de continuation peuvent être utilisées pour décrire le mot prototype.

Pour spécifier une carte de continuation, un caractère différent d'un blanc devra être perforé dans la colonne 72.

Les cartes de séquences sont lues à partir de la colonne 16.

Le nombre maximum de cartes de continuation admis est de 14.

Si le programmeur veut spécifier un paramètre par carte, il le peut, à condition que le premier paramètre apparaisse dans la même carte que le nom de la macro.

Exemple :

1	10	16	72
\$LABEL = ADD			
		\$PAR1 ,	X
		\$PAR2 ,	X
		\$PAR3	

1-4 - Le corps de la macro

Le corps de la macro décrit la séquence d'instructions élémentaires qui constituent sa définition.

Les instructions utilisées pour définir une macro suivent les mêmes règles que les instructions TAS-86, la seule différence étant l'usage de paramètres formels.

Les facilités additionnelles offertes par le macro-générateur comme : appels emboîtés, assemblage conditionnel, variables internes et concaténation, seront discutées séparément.

Un paramètre formel peut être utilisé dans n'importe quelle partie d'une instruction, c'est-à-dire, comme étiquette code opération ou opérande.

Les cartes de continuation ne sont pas admises dans les instructions qui constituent le corps de la macro. Chaque instruction doit être contenue dans une carte.

1-5 L'instruction fin de macro - MEND

De la même manière que l'instruction MACRO précède une définition de macro, l'instruction MEND indique la fin de la définition.

Sa définition est simplement :

<fin de macro> = MEND

1-5 - Usage des commentaires

Des commentaires peuvent s'introduire de la même façon que dans les instructions TAS-86.

Le début d'un commentaire est signalé par une virgule, suivie de deux signes moins ",- -" séparés, au moins, par un blanc, du dernier caractère significatif utilisé dans l'instruction. Les commentaires ne sont pas admis dans le mot prototype.

1-7 - Définitions de macro

En résumant ce qui a été décrit dans les items précédents, une définition de macro a la structure schématique :

```
< instruction MACRO >  
  < mot prototype >  
  { corps de la macro  
    { (instructions TAS-86) }  
  }  
< instruction MEND >
```

Dans le cas de ADD, nous aurons :

```
MACRO
$LABEL = ADD $PAR1, $PAR2, $PAR3
$LABEL = B    $PAR2 , -- charger le 2ème paramètre --
          A    $PAR3 , -- calculer PAR2 + PAR3 --
          C    $PAR1 , -- stocker le résultat en PAR1 --
MEND
```

Prenons comme exemple le cas d'une macro VECTOR qui permet d'effectuer une certaine opération avec le nième élément d'un tableau à une dimension :

```
MACRO
$LABEL = VECTOR $VECT, $N, $COD
LABEL = C      SAVE ,-- préservation de l'accumulateur --
              B    $N ,-- chargement de N--
              AA   $VECT ,-- calcul de l'adresse du nième
                        élément --
              CA   2L ,-- stockage de l'adresse dans
                        * + 2 --
              B    SAVE ,-- restauration de l'accumulateur
              $COD OU ,-- exécution de l'opération
MEND
```

VECT représente le nom du tableau
N représente l'ordre de l'élément.
COD représente l'opération.

Comme nous aurons l'occasion de voir plus tard, lorsque seront traités les appels de macro, si nous désirons ajouter le contenu de l'accumulateur à TABLE (J), nous devons simplement écrire :

```
CALC = *VECTOR TABLE, J, A
```

Des paramètres formels peuvent apparaître dans n'importe quelle partie d'une instruction ; toutes les propositions suivantes sont donc valables :

```
§PAR1 = (§PAR2)           ,-- Réserve de mémoire --
§PAR1 = §PAR2 'PAR3'H      ,-- Utilisation d'une constante
                           hexa-décimale--
§PAR1 = ''§PAR2''          ,-- Définition d'une constante
                           alpha-numérique --
```

Ces possibilités donnent au programmeur une grande souplesse pour définir les macros.

CHAPITRE III

FACILITES ADDITIONNELLES FOURNIES PAR LE MACRO-GENERATEUR

1 - ASSEMBLAGE CONDITIONNEL

1-1 - Définition

Bien que les paramètres formels permettent au programmeur de définir des macros d'une façon très générale, il n'existe aucune méthode capable de donner au macro-générateur la possibilité de générer des séquences différentes d'instructions pour la même macro.

La faculté de décider pendant le temps de génération quelles instructions devront être générées, selon certaines conditions, s'avère un outil fondamental ; elle est connue sous le nom d'Assemblage Conditionnel.

Certaines instructions spéciales permettent de faire générer par le macro-générateur, certaines instructions, en fonction de conditions externes ou internes.

Ces instructions seront décrites dans les paragraphes suivants.

1-2 - Étiquettes internes

Une étiquette interne nomme une instruction dans le corps d'une macro, à laquelle une instruction d'assemblage ira se référer. Ces étiquettes ne sont jamais reproduites par le macro-générateur dans le programme généré.

Elle sont utilisées pour contrôler les branchements internes pendant le temps de génération.

Le format est :

$$\begin{aligned} \langle \text{étiquette int.} \rangle &:: = . \langle \text{nom} \rangle \langle b \rangle = \\ \langle \text{nom} \rangle &:: = \langle \text{lettre} \rangle \left\{ \langle \text{lettre} \rangle \mid \langle \text{chiffre} \rangle \right\}^5_0 \\ \langle \text{lettre} \rangle &:: = A|B \dots |Y|Z \\ \langle \text{chiffre} \rangle &:: = 0|1 \dots |8|9 \\ \langle b \rangle &:: = \text{b} \langle b \rangle | \text{caractère vide} \end{aligned}$$

b représente un blanc -

Exemple :

.LABEL = B \$PAR1

Les étiquettes internes doivent commencer par un point et le **premier** caractère doit être alphabétique.

Un maximum de six caractères (le point non compris) est admis. Si la longueur maximum est dépassée, seuls les six premiers seront pris.

1-3 Variables internes

1-3-1 - Définition

Les variables internes sont des identificateurs déclarés par le programmeur et dont les valeurs peuvent être modifiées durant le temps de génération. Elles peuvent être utilisées comme indicateurs, comme variables auxiliaires, comme compteurs, ou de n'importe quelle manière qui les rende utiles au programmeur. Plusieurs exemples d'applications seront donnés plus tard.

Les variables internes peuvent être divisées en 2 groupes :

- Locales
- et - Globales

Les variables locales ne sont connues que dans la macro où elles sont définies. Dès que la macro est terminée, leurs valeurs se perdent et elles deviennent inaccessibles.

Les variables globales sont connues pendant toute la génération. Elles peuvent être définies dans une certaine macro et être mentionnées dans une autre ou par la même, lors de différents appels.

Les variables internes peuvent avoir deux natures différentes :

- type caractère
- ou - type arithmétique

1-3-2 - Déclaration des variables internes

Avant de se référer à une variable interne, le programmeur est obligé de la déclarer. Il existe quatre instructions de déclaration qui doivent être utilisées dans chaque cas correspondant.

- DCLA - Déclaration de variable arithmétique locale
- DCLC - Déclaration de variable caractère locale
- DCGA - Déclaration de variable arithmétique globale
- DCGC - Déclaration de variable caractère globale.

La définition de l'instruction de déclaration est :

$\langle \text{instruction de décl.} \rangle :: = \langle \text{mot décl.} \rangle \text{ b } \langle \text{b} \rangle \langle \text{liste de variab} \rangle$
 $\langle \text{liste de variables} \rangle :: = \langle \text{nom} \rangle \{ \langle \text{b} \rangle , \langle \text{b} \rangle \langle \text{nom} \rangle \}_0^6$
 $\langle \text{nom} \rangle :: = \langle \text{lettre} \rangle \{ \langle \text{lettre} \rangle | \langle \text{chiffre} \rangle \}_0^5$
 $\langle \text{mot décl.} \rangle :: = \text{DCLA} | \text{DCGA} | \text{DCLC} | \text{DCGC}$
 $\langle \text{lettre} \rangle :: = \text{A} | \dots | \text{Z}$
 $\langle \text{chiffre} \rangle :: = \text{0} | \dots | \text{9}$
 $\langle \text{b} \rangle :: = \text{b } \langle \text{b} \rangle | \text{caractère vide.}$

Une instruction de déclaration peut être utilisée pour chaque variable, mais le programmeur peut définir jusqu'à sept variables dans une instruction, à condition qu'elles soient de la même nature.

Les cartes de continuation ne sont pas admises lorsqu'il s'agit de déclarer des variables internes. Si l'on veut déclarer plus de sept variables d'une même nature, il faudra utiliser des instructions de déclarations additionnelles.

Les considérations formulées ci-dessus s'appliquent également aux 4 types d'instructions de déclaration.

Exemple :

```
DCGC  VAR, CAR
DCLA  COUNT, SWITCH
DCLC  FIELD
DCGA  LOOP
```

Les identificateurs pour une variable interne peuvent avoir jusqu'à six caractères et doivent commencer par une lettre. Dans le cas où la longueur maximum est dépassée seuls les six premiers caractères seront pris en compte.

1-3-3 - Valeurs initiales et Attributs

Toutes les variables sont automatiquement initialisées par le macro générateur quand elles sont déclarées.

Les variables arithmétiques sont initialisées à zero et les variables caractères à des blancs.

La valeur maximum d'une variable arithmétique est $2^{31} - 1$. Valeur maximum d'une variable représentée sous forme binaire, dans l'IBM 360.

Seulement des valeurs entières sont prises en considération. La partie fractionnaire résultant de calculs intermédiaires est ignorée.

Les variables caractères contiennent des chaînes de 1 à 15 caractères.

1-4 - L'instruction SETA

L'instruction SETA est utilisée pour affecter une valeur numérique à une variable interne arithmétique.

L'instruction SETA fourni l'unique moyen d'altérer la valeur d'une variable interne arithmétique.

Toute variable interne, lorsqu'elle est mentionnée dans une instruction, doit être précédée de "§ §".

Ces deux caractères indiquent au macro-générateur qu'une référence à une variable interne est faite.

La définition de l'instruction SETA est :

$\langle \text{inst. SETA} \rangle :: = \langle \text{variable objet} \rangle \langle b \rangle = \langle b \rangle \text{SETA } b \langle b \rangle \langle \text{expr. arith.} \rangle$
 $\langle \text{variable objet} \rangle :: = \text{\$} \text{\$} \langle \text{nom} \rangle$
 $\langle \text{expr. arith.} \rangle :: = \langle \text{opérande} \rangle | \langle \text{opérande} \rangle \langle b \rangle \langle \text{opérateur arith.} \rangle$
 $\langle b \rangle \langle \text{opérande} \rangle$
 $\langle \text{opérande} \rangle :: = \text{\$} \text{\$} \langle \text{nom} \rangle | \text{\$} \langle \text{nom} \rangle | \text{constante numérique}$
 $\langle \text{opérateur arith.} \rangle :: = + | - | * | /$
 $\langle \text{constante numérique} \rangle :: = \{ + | - \} \begin{matrix} 1 \\ 0 \end{matrix} \langle \text{chiffre} \rangle \left\{ \langle \text{chiffre} \rangle \right\}_0^7$
 $\langle \text{nom} \rangle :: = \langle \text{lettre} \rangle \left\{ \langle \text{lettre} \rangle | \langle \text{chiffre} \rangle \right\}_0^5$
 $\langle \text{lettre} \rangle :: = A | B | \dots | Y | Z$
 $\text{chiffre} :: = 0 | 1 | \dots | 9$
 $\langle b \rangle :: = b \langle b \rangle | \text{caractère vide.}$

La variable représentée par $\langle \text{nom} \rangle$ dans $\langle \text{variable objet} \rangle$ a dû être déclarée comme une variable interne arithmétique. L'effet de l'instruction SETA est d'affecter le résultat de l'expression arithmétique à la variable associée à $\langle \text{var. objet} \rangle$

Exemple :

Supposons que deux variables arithmétiques ont été déclarées dans une instruction DCLA précédente

DCLA COUNT, IND

Ces variables peuvent être l'objet d'une instruction SETA

- a) $\text{\$} \text{\$} \text{COUNT} = \text{SETA } \text{\$} \text{\$} \text{IND}$
- b) $\text{\$} \text{\$} \text{COUNT} = \text{SETA } 5$
- c) $\text{\$} \text{\$} \text{COUNT} = \text{SETA } \text{\$} \text{\$} \text{IND} + \text{\$} \text{\$} \text{COUNT}$
- d) $\text{\$} \text{\$} \text{COUNT} = \text{SETA } \text{\$} \text{\$} \text{COUNT} + 1$
- e) $\text{\$} \text{\$} \text{COUNT} = \text{SETA } \text{\$} \text{\$} \text{PAR}$

- En a) la valeur de IND est affectée à COUNT ;
- En b) COUNT prend la valeur d'une constante ;
- En c) le résultat de IND + COUNT est affecté à COUNT ;
- En d) la valeur de COUNT est augmentée de 1 ;
- En e) COUNT prend la valeur d'un paramètre formel.

A chaque fois que la macro est appelée, la valeur effective de PAR est affectée à COUNT.

N'importe quel paramètre (constantes, variables internes arithmétiques, variables internes caractères, paramètres formels) peut être un opérande d'une instruction SETA, comme il a été précisé dans sa définition.

Lorsqu'il ne s'agit pas de variables arithmétiques, le macro-générateur les convertira automatiquement en une valeur numérique. Si la conversion ne peut pas être faite, il y aura erreur.

Dans le cas de e), PAR doit correspondre à une constante numérique, lorsque la macro sera appelée. Dans le cas d'erreur, le macro-générateur ignore l'instruction SETA.

L'expression arithmétique, comme nous l'avons définie, n'admet qu'un maximum de deux opérandes, dans une des formes :

Operande1	+	Operand2
Operande1	-	Operande2
Operande1	*	Operande2
Operande1	/	Operande2

Si des calculs très compliqués doivent s'effectuer - ce qui normalement ne se produira pas - plusieurs instructions SETA pourront être utilisées.

La valeur numérique d'une variable interne arithmétique peut toujours être utilisée comme une variable pour générer des instructions TAS-86. Le macro-générateur les convertira automatiquement en forme caractère. Si, par exemple, COUNT est déclarée comme une variable interne arithmétique, les instructions ci-dessous resulteraient dans l'expansion suivante, admettant COUNT égal à 10 :

```
TABLE = ($$COUNT) , -- Réserve --  
      B $COUNT , -- Génération d'un littéral --  
      C VAR + $$COUNT
```

```
TABLE = (10) , -- Réserve --  
      B 10 , -- Génération d'un littéral --  
      C VAR + 10
```

1-5 - La concaténation

Définition :

$\langle \text{concaténation} \rangle :: = \langle \text{opérande} \rangle \langle \text{opérateur de concat.} \rangle \langle \text{opérande} \rangle$
 $\langle \text{opérande} \rangle :: = \$\$ \langle \text{nom} \rangle \mid \$ \langle \text{nom} \rangle \mid \langle \text{chaîne de caract.} \rangle$
 $\langle \text{nom} \rangle :: = \langle \text{lettre} \rangle \left\{ \langle \text{lettre} \rangle \mid \langle \text{chiffre} \rangle \right\}_0^5$
 $\langle \text{chaîne de caract.} \rangle :: = \left\{ \langle \text{chiffre} \rangle \mid \langle \text{lettre} \rangle \right\}_1^{15}$
 $\langle \text{opérateur de concat.} \rangle :: = \&$
lettre :: = A | ... | Z
chiffre :: = 0 | ... | 9

L'effet de la concaténation est d'aligner le 2ème opérande immédiatement après le premier.

Dans des circonstances spéciales il peut être utile de former les noms des variables en faisant la concaténation de deux champs différents.

Supposons que le programmeur souhaite générer des étiquettes par la concaténation d'une variable formelle avec une variable interne arithmétique qui serait augmentée de 1, après chaque génération d'étiquette.

```
LAB & COUNT = B ASAVE  
COUNT = SETA COUNT + 1
```

Si une boucle interne s'établit, en utilisant des instructions de branchement interne, comme nous le verrons plus tard, à chaque fois le macro-générateur va générer une étiquette différente pour la première instruction. Si, par exemple, LAB est remplacé par la variable JUMP, quand la macro est appelée, nous aurons des instructions générées à chaque fois, comme :

```
JUMPO = B ASAVE  
JUMP1 = B ASAVE  
etc ...
```

Dans l'opération de concaténation, il ne doit pas exister de blancs séparant les variables et l'opérateur, comme il a été précisé dans sa définition.

Ainsi, les deux instructions ci-dessous ne sont pas valables :

```
LAB & COUNT = B ASAVE  
LAB & COUNT = B ASAVE
```

Une concaténation peut être effectuée dans n'importe quel partie d'une instruction (étiquette, code opération et opérande).

Par suite de cette généralité toutes les instructions qui suivent sont valables.

- a) NAME = B&INT SAVE
- b) NAME&LAB = B VAR&COUNT
- c) CONST = ' ' TITLE&INT ' '

En a) nous avons fait la concaténation d'une variable interne (que nous admettons du type caractère) avec B, pour générer un code d'opération. Si, par exemple, INT était égal à E, nous aurions généré BE. Si INT était un blanc, seulement B serait généré.

En b) la concaténation est employée pour modifier les parties étiquette et opérande.

En c) elle est employée pour construire une constante alphanumérique.

Toutes les combinaisons possibles sont admises pour les deux Opérandes, dans une concaténation, sauf dans le cas d'une concaténation de deux constantes. Au moins un des opérandes devra être, soit un paramètre formel, soit une variable interne.

Le cas de la concaténation de deux constantes est considérée comme non valable, dû au fait qu'elle provoque un effort inutile de la part du macro-générateur.

Exemples :

- a) &VAR&1 = (20)
- b) TAB&COUNT = (10)
- c) TAB&1 = (20)

a) et b) représentent des concaténations valables mais c) serait accusée comme une erreur. Si en b), COUNT est une variable interne arithmétique, elle sera automatiquement convertie en forme caractère, cadrée à gauche. Dans ce cas, si la valeur de COUNT était 6, l'instruction générée serait TAB6 = (10)

1-6 - L'instruction SETC

1-6-1 - Définition

L'instruction SETC est utilisée pour manipuler les variables internes caractères, de la même façon que l'instruction SETA manipule les variables arithmétiques. Elle est utilisée pour altérer la valeur d'une variable interne caractère.

Son format est :

$\langle \text{instruction SETC} \rangle :: = \langle \text{var. objet} \rangle \langle b \rangle = \langle b \rangle \text{ SETC } \langle b \rangle$
 $\langle \text{expression} \rangle$

$\langle \text{var. objet} \rangle :: = \$\$ \langle \text{nom} \rangle$

$\langle \text{expression} \rangle :: = \langle \text{opérande caractère} \rangle \langle \text{opérande caract.} \rangle \&$
 $\langle \text{opérande caract.} \rangle | \langle \text{instruction move.} \rangle$

$\langle \text{opérande caract.} \rangle :: = \$ \langle \text{nom} \rangle | \$\$ \langle \text{nom} \rangle | \langle \text{constante} \rangle | \langle \text{carac-} \rangle$
 $\text{tère spécial} \rangle$

$\langle \text{constante} \rangle :: = \left\{ \langle \text{lettre} \rangle | \langle \text{chiffre} \rangle \right\}_1^6$

$\langle \text{instruction move} \rangle :: = \text{MOVE}(\langle \text{opérande source} \rangle, \langle \text{opérande de} \rangle$
 $\text{position} \rangle, \langle \text{opérande de position} \rangle,$
 $\text{opérande de position})$

$\langle \text{opérande source} \rangle :: = \$\$ \langle \text{nom} \rangle | \$ \langle \text{nom} \rangle | \left\{ \langle \text{lettre} \rangle | \langle \text{chiffre} \rangle \right\}_1^{15}$

$\langle \text{nom} \rangle :: = \langle \text{lettre} \rangle \left\{ \langle \text{lettre} \rangle | \langle \text{chiffre} \rangle \right\}_0^5$

$\langle \text{opérande de position} \rangle :: = \$\$ \langle \text{nom} \rangle | \$ \langle \text{nom} \rangle | \left\{ \langle \text{chiffre} \rangle \right\}_1^2$

$\langle \text{lettre} \rangle :: = A | B | \dots | Z$

$\langle \text{chiffre} \rangle :: = 0 | 1 | \dots | 9$

$\langle \text{caractère spécial} \rangle :: = \& | + | - | / | , | \& | \cdot | * | . | \$ | :$

$\langle b \rangle :: = \text{caractère vide} | \& \langle b \rangle$

La variable représentée par <nom>en<variable objet>a dû être déclarée comme une variable interne caractère. L'effet de l'instruction SETC est d'affecter le résultat de l'expression à la variable associée à <variable objet.>

La signification des différents opérandes figurant dans l'expression sera expliquée dans les prochains items.

1-6-2 - L'expression

Comme il a été précisé dans la définition, l'expression peut avoir une des formes suivantes (CHAR est déclaré comme une variable interne caractère) :

a) %CHAR = SETC opérande

L'opérande peut être un paramètre formel, une variable interne ou une constante; Quand il s'agit d'une variable interne arithmétique elle est automatiquement convertie en caractère, justifiée à gauche et complétée avec des blancs à la droite, si nécessaire.

L'effet de l'instruction SETC est d'affecter la valeur de l'opérande, sous la forme caractère, à la variable objet spécifiée dans la partie étiquette. Si l'opérande a moins de 15 caractères, il est complété à droite par des blancs, avant l'affectation.

Exemple :

```
%CHAR = SETC %PAR  
%CHAR = SETC TEST  
%CHAR = SETC %ARIT
```

Dans le premier cas, la valeur effective de PAR est affectée à CHAR, lorsque la macro est appelée.

Dans le deuxième cas, la chaîne TEST est affectée à CHAR. Comme TEST a 4 caractères, nous aurons 11 blancs de complément à droite. Une constante sous forme caractère ne peut pas avoir plus de 6 caractères, dû aux restrictions TAS-86.

Dans le troisième cas, supposant que ARIT est une variable interne arithmétique, sa valeur est convertie à la forme caractère et affectée à CHAR.

Si ARIT était égal à 15, après l'exécution de l'instruction SETC, CHAR deviendrait 15bbbbbbbbbbbbbb (b représente un blanc).

b) $\$CHAR = SETC$ 'caractère spécial'

Cette forme est employée pour affecter un caractère spécial à la valeur de CHAR.

Exemple :

```
 $\$CHAR = SETC$  '+'  
 $\$CHAR = SETC$  ' '
```

Le caractère doit apparaître entre deux apostrophes.

Dans ce cas, un seul caractère est permis, dans la partie opérande. Après l'exécution de l'instruction SETC, pour le premier cas, CHAR sera égal à :

+bbbbbbbbbbbbbb

C) ~~\$\$~~CHAR = SETC opérande1 & opérande2

Dans ce cas, l'expression consiste en une concaténation. Les opérandes 1 et 2 peuvent être, soit une variable formelle, soit une variable interne, soit une constante, soit un caractère spécial.

Exemples:

```
$$CHAR = SETC $PAR&'.'
```

```
$$CHAR = SETC $$CHAR&12
```

```
$$CHAR = SETC $$CHAR&SLAB&'.'
```

```
$$CHAR = SETC STRING&12
```

Le troisième cas ne serait pas valable à cause des règles de la concaténation (plus de deux opérandes). Le dernier exemple ne serait également pas valable pour la même raison (concaténation de deux constantes).

d) ~~\$\$~~CHAR = SETC OPERATION MOVE

L'opération MOVE offre la moyen de transférer chaînes ou sous-chaînes de caractères d'un champ à une variable interne caractère.

L'opération MOVE est spécifiée de la façon suivante :

MOVE (opérande1, opérande2, opérande3, opérande4)

~ opérande1

Il spécifie la chaîne source et peut être, soit une variable formelle, soit une constante, soit une variable interne caractère. L'utilisation d'une variable interne arithmétique spécifiée comme chaîne source serait détectée comme une erreur.

- opérande 2

Il indique la position, dans l'opérande1, du premier caractère à être transféré. Il peut être soit un paramètre formel, soit une constante numérique, soit une variable interne arithmétique.

- opérande 3

Il précise le nombre de caractères à être transférés de l'opérande1. Il peut être, soit un paramètre formel, soit une constante numérique, soit une variable interne arithmétique.

- opérande4

Il indique la première position dans <variable objet> celle-ci est la variable interne caractère dans la partie étiquette de l'instruction SETC, ou les caractères déplacés de la chaîne source seront insérés. Il a la même nature que les opérandes 2 et 3.

Les variables internes caractères ne peuvent pas être utilisées pour représenter les opérandes 2,3,4.

Exemple :

```
%%CHAR = SETC MOVE(%PAR,1,4,%%ARIT)
```

Si CHAR contient des blancs avant l'opération et si

PAR = AF806 et ARIT = 3 , après l'opération MOVE, CHAR deviendra :

66AF806666666666

1-7 - L'instruction de branchement inconditionnel (UAB)

Sa définition est :

$$\langle \text{instruction UAB} \rangle :: = \left\{ \langle \text{étiquette interne} \rangle \langle b \rangle \right\}_0^1 \text{ UAB.}$$

$$\begin{aligned} & \text{b} \langle b \rangle \text{ nom d'instruction} \\ \langle \text{nom d'instruction} \rangle & :: = . \langle \text{lettre} \rangle \left\{ \langle \text{lettre} \rangle \mid \langle \text{chiffre} \rangle \right\}_0^5 \end{aligned}$$

$$\begin{aligned} \langle \text{étiquette int.} \rangle & :: = . \langle \text{nom} \rangle \langle b \rangle = \\ \langle \text{nom} \rangle & :: = \langle \text{lettre} \rangle \left\{ \langle \text{lettre} \rangle \mid \langle \text{chiffre} \rangle \right\}_0^5 \end{aligned}$$

$$\langle b \rangle :: = \text{b} \langle b \rangle \mid \text{caractère vide}$$

$$\langle \text{lettre} \rangle :: = A \mid B \mid \dots \mid Y \mid Z$$

$$\langle \text{chiffre} \rangle :: = 0 \mid 1 \mid \dots \mid 9$$

Cette instruction produira un branchement incondi-
tionnel au moment de la génération à l'instruction nommée $\langle \text{nom}$
d'instruction $\rangle$. Il est évident que nom d'instruction se réfère
à une instruction nommée par une étiquette, quelque part
ailleurs, dans le programme.

Exemple :

```

B      SAVE
A      ⌫PAR1
UAB    .LABEL
ML     ⌫PAR2
SH     ZL 23
.LABEL = C  SAVE

```

En supposant que ces instructions fassent partie
d'une certaine macro, et que PAR1 et PAR2 doivent être
remplacés par les paramètres effectifs X et Y, le code généré
serait :

```

B      SAVE
A      X
-      -

```

Dans ce cas particulier, les instructions de multiplication et de décalage ne seraient jamais générées, par suite de l'instruction UAB. Il faut bien expliciter que des instructions spéciales, telles que les déclarations, SETA, SETC, branchement conditionnel (décrit dans le prochain paragraphe) et incondi- tionnel, servent uniquement à spécifier certaines opérations au macro-générateur et ne sont jamais insérées dans le code final résultant.

1-8 L'instruction de branchement conditionnel (CAB)

Sa définition est :

$$\begin{aligned} \langle \text{instruction CAB} \rangle &:: = \left\{ \langle \text{étiquette int.} \rangle \langle b \rangle \right\}_0^1 \text{ CAB } \bar{b} \langle b \rangle \\ &\quad \langle \text{condition} \rangle \langle b \rangle \langle \text{nom d'instruction} \rangle \\ \langle \text{condition} \rangle &:: = \langle \text{opérande1} \rangle \langle b \rangle \langle \text{opérateur de comparaison} \rangle \\ &\quad \bar{b} \langle b \rangle \\ \langle \text{opérande2} \rangle & \\ \langle \text{opérande1} \rangle &:: = \mathcal{S} \mathcal{S} \langle \text{nom} \rangle | \mathcal{S} \langle \text{nom} \rangle | \langle \text{constante} \rangle | \langle \text{constante numé.} \rangle \\ \langle \text{opérande2} \rangle &:: = \langle \text{opérande1} \rangle | \langle \text{caractère spécial} \rangle \\ \langle \text{opérateur de comparaison} \rangle &:: = \text{EQ} | \text{NE} | \text{GT} | \text{GE} | \text{LT} | \text{LE} | \text{NG} | \text{NL} \\ \langle \text{étiquette int.} \rangle &:: = . \langle \text{nom} \rangle \langle b \rangle = \\ \langle \text{nom d'instruction} \rangle &:: = . \langle \text{nom} \rangle \\ \langle \text{nom} \rangle &:: = \langle \text{lettre} \rangle \left\{ \langle \text{lettre} \rangle | \langle \text{chiffre} \rangle \right\}_0^5 \\ \langle \text{constante numérique} \rangle &:: = \langle \text{chiffre} \rangle \left\{ \langle \text{chiffre} \rangle \right\}_0^7 \\ \langle \text{lettre} \rangle &:: = \text{A} | \text{B} | \dots | \text{Z} \\ \langle \text{chiffre} \rangle &:: = \text{0} | \dots | \text{9} \\ \langle \text{caractère spécial} \rangle &:: = \bar{b} | + | - | / | , | ' | . | : | = | \& | \mathcal{S} \\ \langle b \rangle &:: = \bar{b} \langle b \rangle | \text{caractère vide} \\ \langle \text{constante} \rangle &:: = \langle \text{lettre} \rangle | \langle \text{chiffre} \rangle \left\} \right\}_1^7 \end{aligned}$$

Cette instruction est utilisée pour tester une certaine condition qui est indiquée entre parenthèses dans la partie opérande de l'instruction CAB. Si la condition se vérifie, un branchement interne est effectué à l'instruction nommée $\langle \text{nom d'instruction} \rangle$, sinon la génération continue en séquence.

Exemple :

```
CAB    ($VAR EQ ' ') .JUMP
B      VAR1
.JUMP =
```

Dans cet exemple, le paramètre formel correspondant à VAR est comparé à un blanc. Si la condition se vérifie, la génération s'effectue à l'instruction nommée JUMP, sinon, la prochaine instruction B VAR1 sera générée.

Les éléments qui constituent le champ opérande sont :

- Opérande1

Il peut être, soit un paramètre formel, soit une variable interne, soit une constante, soit une constante numérique.

- Opérateur de comparaison

Il détermine le test qui doit être effectué.

Les opérateurs valables sont :

EQ - égal

NE - inégal (non égal)

GT - plus grand

GE - plus grand ou égal

LT - moins que

LE - moins ou égal

NG - non supérieur

NL - non inférieur

- Opérande2

Il peut être, soit un paramètre formel, soit une variable interne, soit une constante, soit une constante numérique, soit un caractère spécial entre apostrophes.

Toutes les combinaisons pour les deux opérandes sont valables, sauf les évidentes. Si une instruction CAB est

employée pour comparer deux constantes, elle sera considérée comme non valable.

Une deuxième restriction concerne l'usage de variables internes caractères dans des comparaisons numériques. Pour ces cas, une **conversion** à la forme numérique ne s'effectue pas et une erreur est **signalée**.

Exemples :

```
a - CAB (VAR GT 5). LABEL
b - CAB (COUNT LT 2). LABEL
c - CAB (CHAR EQ '+'). LABEL
d - CAB (CHAR GT 5). LABEL
e - CAB (VAR EQ ' '). LABEL
f - CAB (STORE EQ ST). LABEL
```

En a - la valeur effective du paramètre VAR sera comparée à 5, lorsque la macro sera appelée.

En b - COUNT (vraisemblablement une variable interne arithmétique) est comparée à la constante 2. Si COUNT est déclarée comme une variable interne caractère, une erreur sera accusée.

En c - CHAR (une variable interne caractère) est comparée au caractère +.

En d - une variable interne caractère est utilisée dans une comparaison numérique, ce qui provoquera une instruction non valable (nous supposons que CHAR a été déclarée comme une variable caractère).

En e - la valeur effective du paramètre VAR est comparée à un blanc.

En f - l'instruction est mauvaise car une comparaison évidente est effectuée.

Dans tous les cas, si la condition formulée entre parenthèses se vérifie, un branchement sera effectué à l'instruction nommée LABEL.

1-9 - L'instruction non-opération (NOP)

La définition est :

$$\begin{aligned} \langle \text{instruction NOP} \rangle &:: = \left\{ \langle \text{étiquette int.} \rangle \right\}_0^1 \langle b \rangle \text{ NOP} \\ \langle \text{étiquette int.} \rangle &:: = . \langle \text{nom} \rangle \langle b \rangle = \\ \langle \text{nom} \rangle &:: = \langle \text{lettre} \rangle \langle \text{lettre} \rangle | \langle \text{chiffre} \rangle \left. \vphantom{\langle \text{nom} \rangle} \right\}_0^5 \\ \langle \text{lettre} \rangle &:: = A | B | \dots | Z \\ \langle \text{chiffre} \rangle &:: = 0 | \dots | 9 \\ \langle b \rangle &:: = \bar{b} \langle b \rangle | \text{caractère vide} \end{aligned}$$

Cette instruction produit une non-opération, ce qui signifie qu'aucune action ne sera prise par le Macro-Générateur. Son intérêt est de fournir un point de branchement pour une instruction CAB ou UAB.

Exemple :

```
UAB      .LABEL
.LABEL = NOP
$COUNT = SETA $COUNT+1
```

Si le point de branchement se situe dans une instruction SETA ou SETC, il n'existe aucun moyen de la nommer car la partie étiquette est occupée par la variable objet.

Dans ces cas une référence peut être faite à une instruction NOP placée immédiatement avant le point de branchement.

1-10 L'instruction NOTE

Sa définition est :

$$\begin{aligned} \langle \text{instruction NOTE} \rangle &:: = \left\{ \langle \text{étiquette int.} \rangle \right\}_0^1 \langle b \rangle \text{NOTE } \bar{b} \langle b \rangle \\ \langle \text{message} \rangle & \\ \langle \text{étiquette int.} \rangle &:: = . \langle \text{nom} \rangle \langle b \rangle = \\ \langle \text{message} \rangle &:: = \left\{ \langle \text{caractère} \rangle \right\}_0^5 \\ \langle \text{nom} \rangle &:: = \langle \text{lettre} \rangle \left\{ \langle \text{lettre} \rangle | \langle \text{chiffre} \rangle \right\}_0^5 \\ \langle \text{lettre} \rangle &:: = A | \dots | Z \\ \langle \text{chiffre} \rangle &:: = 0 | 1 | \dots | 9 \\ \langle \text{caractère} \rangle &:: = \langle \text{lettre} \rangle | \langle \text{chiffre} \rangle | * | \bar{b} | + | - | / | \emptyset | , | . | : | ; | ' | ! | \& = \\ \langle b \rangle &:: = \bar{b} \langle b \rangle | \text{caractère vide} \end{aligned}$$

L'effet de NOTE est simplement d'imprimer le contenu du message, qui peut contenir jusqu'à 50 caractères. Cette instruction peut être utilisée pour générer des diagnostics produits par les macros, lorsqu'une condition quelconque n'est pas satisfaite par les paramètres effectifs. L'impression apparaît ainsi :

WARNING _contenu du message.

1-11 - L'instruction de non perforation (NPUN)

Définition :

$$\begin{aligned} \langle \text{instruction NPUN} \rangle &:: = \left\{ \langle \text{étiquette int.} \rangle \right\}_0^1 \langle b \rangle \text{NPUN} \\ \langle \text{étiquette int.} \rangle &:: = . \langle \text{nom} \rangle \langle b \rangle = \\ \langle \text{nom} \rangle &:: = \langle \text{lettre} \rangle \left\{ \langle \text{lettre} \rangle | \langle \text{chiffre} \rangle \right\}_0^5 \\ \langle \text{lettre} \rangle &:: = A | \dots | Z \\ \langle \text{chiffre} \rangle &:: = 0 | \dots | 9 \\ \langle b \rangle &:: = \bar{b} \langle b \rangle | \text{caractère vide.} \end{aligned}$$

L'instruction NPUN s'explique par le fait que le macro-générateur, à la fin de la traduction, met le programme généré sur bande magnétique ou sur cartes perforées ; ce fichier de sortie sera lu à la deuxième phase du processus par la machine Téléfunken, comme il a été expliqué dans la 1ère partie. Cependant, si durant la génération une grave erreur se produit, le programme généré sera certainement faux. Dans ce cas, afin de prévenir toute opération de sortie inutile, l'instruction NPUN peut être utilisée. Sa fonction sera d'empêcher l'écriture du programme générée sur périphérique.

De toute manière, une sortie sera produite sur imprimante.

CHAPITRE IV

APPELS DE MACROS

1 - DEFINITION

$\langle \text{macro appel} \rangle : : = \{ \langle \text{identificateur} \rangle \langle b \rangle \}^1 \langle b \rangle * \langle \text{nom} \rangle \{ \langle b \rangle \}$
 $\{ \langle \text{liste d'arguments} \rangle \}^1_0$
 $\langle \text{identificateur} \rangle : : = \langle \text{nom} \rangle \{ \langle \text{point} \rangle \}^1_0$
 $\langle \text{nom} \rangle : : = \langle \text{lettre} \rangle \{ \langle \text{lettre} \rangle \langle \text{chiffre} \rangle \}^5_0$
 $\langle \text{point} \rangle : : = .$
 $\langle \text{liste d'arguments} \rangle : : = \langle \text{paramètre effectif} \rangle \{ \langle b \rangle \}^1_0$
 $\langle \text{paramètre effectif} \rangle : : = \{ \langle \text{symbole} \rangle \{ \langle \text{caractère} \rangle \}^7_1 \}$
 $\langle \text{lettre} \rangle : : = A | \dots | Y | Z$
 $\langle \text{chiffre} \rangle : : = 0 | \dots | 9$
 $\langle \text{symbole} \rangle : : = \langle \text{lettre} \rangle | \langle \text{chiffre} \rangle | + | - | * | / | . | \% | = | \&$
 $\langle \text{caractère} \rangle : : = \langle \text{symbole} \rangle \{ \langle b \rangle \}^1_0$
 $\langle b \rangle : : = \langle b \rangle | \text{caractère vide}$

Dans les chapîtres précédents nous avons décrit les possibilités offertes par le Macro-Générateur, en ce qui concerne la définition de macros. Une fois définie, la macro peut être mentionnée dans une instruction du macro-langage.

Pour pouvoir appeler une macro, le programmeur écrira simplement son nom, immédiatement précédé d'un astérisque. Dans la partie opérande se trouve une liste d'arguments, séparée par des virgules, qui spécifie les paramètres effectifs.

Comme il a été précisé dans sa définition, la liste d'arguments accepte également que des caractères spéciaux en fassent partie. Cette possibilité devient utile dans quelques cas particuliers, que nous aurons l'occasion d'examiner à la fin de ce chapitre. La partie étiquette peut éventuellement contenir une étiquette effective qui nommera l'instruction correspondante indiquée dans la

définition de la macro.

Exemple :

```
MACRO
$LAB = ADD $PAR1, $PAR2, $PAR3
      B  $PAR1
      A  $PAR2
$LAB = C  $PAR3
MEND
```

Pour appeler cette macro, le programmeur écrira simplement :

```
MAC1 = *ADD Y,X,Z
```

qui produira le code généré suivant :

```
B      Y
A      X
MAC1 = C      Z
```

2. La liste d'arguments

Dans la majorité des cas, une correspondance directe s'établit entre la liste de paramètres formels et celle des paramètres effectifs étant donné qu'elles contiennent le même nombre d'éléments. Les paramètres formels sont alors remplacés par les effectifs figurant dans la position correspondante de la liste d'arguments, celle-ci étant spécifiée dans la partie opérande de l'instruction d'appel.

Dans l'exemple précédent, Y était associé à PAR1, X à PAR2 et Z à PAR3.

Si le programmeur souhaite préciser qu'un certain paramètre est absent et doit être remplacé par des blancs, une virgule doit apparaître à la place du paramètre manquant.

Dans le cas précédent, si nous voulons préciser que X est absent, nous devons écrire :

```
MAC1 = *ADD Y,,Z
```

Il est à remarquer que cela mènerait le macro-générateur à générer une instruction TAS-86 non valable (A 555)!

Cette éventualité peut être facilement évitée par l'usage d'instructions d'assemblage conditionnel dans la définition de ADD :

```
MACRO
$LAB = ADD    $PAR1,$PAR2,$PAR3
        B      $PAR1
        CAB    ($PAR2 EQ ' ').L1
        A      $PAR2
.L1 = NOP
LAB = C      $PAR3
MEND
```

Avec cette nouvelle définition, l'appel

```
MAC1 = *ADD Y,,Z
```

produira le code généré suivant :

```
        B      Y
MAC1 =  C      Z
```

qui est une simple affectation.

Une virgule supplémentaire est requise dans cet exemple car le paramètre manquant est placé au milieu de la liste.

Si des paramètres manquants ne sont pas entourés par d'autres présents, les virgules supplémentaires peuvent être ignorées.

Dans notre cas, si PAR2 et PAR3 étaient remplacés par des blancs, l'instruction d'appel serait simplement :

```
MAC1 = *ADD X
```

Les deux derniers paramètres sont automatiquement considérés comme des blancs, par le macro-générateur.

Un paramètre effectif peut contenir un maximum de 7 caractères, comme nous avons vu dans la définition de <liste d'arguments>.

3. Cartes de continuation

Les mêmes facilités décrites pour le mot prototype s'appliquent à l'instruction d'appel.

Des cartes de continuation peuvent être utilisées pour contenir la liste d'arguments. Afin de signaler qu'une carte de séquence suit, un caractère différent d'un blanc devra être perforé à la colonne 72. Lorsqu'il s'agit d'une carte de continuation, le macro-générateur commence sa lecture à partir de la colonne 16. Il est permis un nombre maximum de 14 de ces cartes.

De la même manière que pour le mot prototype, les blancs sont toujours ignorés.

Prenons l'exemple de l'appel d'une certaine macro ASSIGN, qui - nous le supposons - est définie avec 15 paramètres :

```

                                72
LABEL = *ASSIGN    ARGUM1,+,ARGUM3, ,    X
                                72
                                ARGUM4
```

Le premier argument est ARGUM1, le deuxième est le caractère **,+**, le troisième ARGUM3, le quatrième des blancs **,**, et finalement ARGUM4. Les 10 derniers arguments sont considérés comme étant des blancs.

La définition de ASSIGN sera envisagée à la fin du chapitre.

4. Caractères spéciaux comme paramètres

Le macro-générateur accepte des caractères spéciaux comme **'+' , '-'** , etc... utilisés comme des paramètres effectifs dans un appel. Cette facilité peut devenir utile, comme nous l'avons dit, dans des cas spéciaux.

Comme les blancs et les virgules sont des unités syntatiques qui précisent la fin d'un argument, si le programmeur désire spécifier un paramètre avec des blancs ou des virgules en faisant partie, ils doivent apparaître entre apostrophes.

Si la chaîne **'ND 5,'** doit être passée comme un paramètre et si une macro END, par exemple, est appelée, nous l'écrirons ainsi :

```
*END    'ND5,',ARG2
```

Si une apostrophe fait, elle-même, partie de l'argument, elle sera représentée par deux, et, s'il existe deux apostrophes effectives à représenter, il en apparaîtra trois. Le macro-générateur n'accepte pas plus de deux apostrophes consécutives.

Il s'agit, bien entendu, d'apostrophes effectives.

Prenons l'exemple d'une macro BUILD appelée avec deux paramètres effectifs **'A2'H** et **"ST"**. De tels paramètres pourraient éventuellement être utilisés comme des constantes hexa-décimales et alpha-numériques. L'instruction d'appel serait :

```
*BUILD    ''A2''H','ST''''
```

De toute manière, le nombre maximum de caractères pour un paramètre effectif est toujours de 7.

Le programmeur doit se rappeler que la duplication d'apostrophes n'est qu'une indication pour le macro-générateur et n'augmente

pas le nombre de caractères effectifs.

5 - EXEMPLE D'APPLICATION

Nous prendrons comme exemple la définition de la macro ASSIGN qui analyse les expressions arithmétiques.

```

MACRO
%LAB = ASSIGN %RES, %V, %OP1, %V1, %OP2,    X
          %V2, %OP3, %V3, %OP4, %V4,    X
          %OP5, %V5, %OP6, %V6

      DCLA    COUNT
      DCLC    VAR,  OP
%LAB      = B    %V
.L1      = NOP
%%COUNT = SETA    %%COUNT+1
          CAB      (%%COUNT GT 1).L2
          CAB      (%OP1 EQ ' ').STORE
%%OP      = SETC    %OP1 ,-- SET OP EQUAL TO OP1--
%%VAR     = SETC    %V1 ,-- SET VAR EQUAL TO V1
          UAB      .SEAROP ,--BRANCH TO SEARCH OPERATOR--
.L2      = CAB      (%%COUNT GT 2).L3
          CAB      (%OP2 EQ ' ').STORE
%%VAR     = SETC    %V2
%%OP      = SETC    %OP2
          UAB      .SEAROP
.L3      = CAB      (%%COUNT GT 3).L4
          CAB      (%OP3 EQ ' ').STORE
%%OP      = SETC    %OP3
%%VAR     = SETC    %V3
          UAB      .SEAROP
.L4      = CAB      (%%COUNT GT 4).L5
          CAB      (%OP4 EQ ' ').STORE
%%OP      = SETC    %OP4
%%VAR     = SETC    %V4
          UAB      .SEAROP

```

```

.L5      = CAB      ($$COUNT GT 5).L6
          CAB      ($OP5 EQ ' ').STORE
$$OP     = SETC     $OP5
$$VAR    = SETC     $V5
          UAB      .SEAROP
.L6      = CAB      ($$COUNT NE 6).STORE
          CAB      (STOP6 EQ ' ') .STORE
$$OP     = SETC     $OP6
$$VAR    = SETC     $V6
.SEAROP  = CAB      ($$OP NE '+').MINUS
          A        $$VAR
          UAB      .L1
.MINUS   = CAB      ($$OP NE '-').MULT
          SB        $$VAR
          UAB      .L1
.MULT    = CAB      ($$OP NE '*').DIV
          ML        $$VAR
          SH        ZL 23
          UAB      .L1
.DIV     = CAB      ($$OP NE '/').ERR
          SU        DIVIDE.
          AA        $$VAR
          UAB      .L1
.ERR     = NOTE     INVALID OPERATOR IN ARGUMENT LIST
          NPUN
          UAB      .L1
.STORE   = C        $RES
          MEND

```

Cette macro permet que l'on écrive des expressions arithmétiques sous une forme qui ressemble de bien près à un langage évolué.

RES représente la variable objet, les variables V les opérandes et la variable OP les opérateurs.

Un appel du type

EXPR = *ASSIGN Y,A,+,B,-,C

produira le code

```
EXPR =  B      A
        A      B
        SB     C
        C      Y
```

La différence fondamentale entre l'analyse effectuée par cette macro et celle d'un compilateur est que ce dernier tient compte de la priorité des opérateurs. La macro génère des opérations selon l'apparition des opérandes, de gauche à droite. Cette limitation est commune **aux traducteurs** proche d'un langage assembleur comme **par exemple le compilateur PL-360.**

Le nombre d'arguments est variable et à chaque fois que le macro-générateur détecte que le prochain argument est un blanc, il génère l'instruction de stockage et termine l'analyse.

Sa forme la plus simple serait :

EXPR= *ASSIGN Y,X

Cet appel génère simplement

```
EXPR = B      X
        C      Y
```

COUNT est une variable interne arithmétique qui va évoluer et indiquer lequel parmi les opérateurs sera analysé.

VAR est une variable interne caractère qui est faite égale à la valeur de l'opérande correspondant, spécifié par un paramètre formel.

A chaque fois qu'un opérateur est détecté, un groupe d'instructions correspondantes est généré et l'analyse se poursuit avec le prochain opérande. Dans le cas où un opérateur non valable est reconnu, un message est imprimé, une instruction de non perforation **est reconnue** et l'analyse se poursuit avec le prochain opérande.

Le lecteur devra essayer de suivre la simple logique de cette macro et de l'appliquer à plusieurs exemples.

DIVIDE est le nom d'un sous-programme qui effectue la division. Il est à remarquer qu'une expression comme :

$Y = (A+B*C)/D$, devra s'écrire :

$Y = (B*C+A)/D$, pour être calculée par l'appel

ASSIGN Y,B,*,C,+,A,/,D

Changer l'ordre des opérandes et, dans quelques cas rares, utiliser des appels consécutifs, rendra possible au programmeur de traiter des expressions arithmétiques compliquées.

CHAPITRE V

APPELS EMBOUTES

1 - DEFINITION

Nous avons décrit dans les trois premiers chapitres les règles de définition et d'appel de macros. Comme nous l'avons déjà vu, dans les définitions, une macro n'appelle jamais une autre dans sa définition. Cette facilité qu'a une macro de générer automatiquement un appel à une autre est connue sous le nom d'emboîtement.

Prenons comme exemple le cas d'une macro SQRE pour calculer le carré d'une certaine variable :

```
MACRO
$LAB = SQRE      $PAR,$RES,$IND
$LAB = B         $PAR
ML      $PAR
SH      ZL 23
CAB      ($IND EQ ' ').END
C      $RES
.END = MEND
```

IND est un indicateur utilisé pour décider quand le résultat devra ou non être stocké en RES. Lorsqu'il s'agit d'un blanc, le résultat est gardé dans l'accumulateur.

Si nous voulons calculer $Y = X^2$, nous écrirons simplement :

```
NAME = *SQRE      X,Y,ST
qui produira
NAME = B          X
ML      X
SH      ZL 23
C      Y
```

Il est à remarquer que, pour provoquer la génération de l'instruction C Y, le troisième argument peut être

n'importe quoi sauf un blanc.

Si, pour calculer des expressions de la forme $Z = X^2 + Y^2$, une autre macro ADDSQR doit être écrite, au lieu de réécrire les instructions pour calculer les carrés, nous pouvons appeler la macro SQRE, lorsque ADDSQR est définie.

De cette manière, nous pourrions définir ADDSQR comme :

```
MACRO
$LAB = ADDSQR    $PAR1,$PAR2,$RES
$LAB = *SQRE     $PAR1,$PAR1,ST
SQUR2 = *SQRE    $PAR2
      A         $PAR1
      C         $RES
MEND
```

Pour calculer $Z = X^2 + Y^2$, nous écrirons simplement :

```
NAME = *ADDSQR   X,Y,Z
qui sera remplacé par :
NAME = B   X
      ML   X
      SH   ZL 23
      C   X
SQUR2= B   Y
      ML   Y
      SH   ZL 23
      A   X
      C   Z
```

Cet exemple donne une idée du système modulaire qui peut être construit en utilisant des appels emboîtés.

Des macros plus compliquées peuvent être définies en utilisant d'autres plus simples qui auront pu, elles-mêmes, être définies par l'appel à d'autres encore et ainsi de suite.

Il existe une limite dans le niveau d'emboîtement, ce qui sera discuté dans les prochains items.

2 - PARAMETRES DES APPELS EMBOITES

Les paramètres spécifiés pour la macro interne (la macro qui est appelée par l'externe) peuvent être mélangés, c'est à dire, soit formels, soit effectifs. S'ils sont formels, ils doivent être inclus dans la liste de paramètres de la macro externe. Toutes ces règles seront plus facilement comprises avec les exemples ci-dessous cités.

Ces exemples ont été choisis uniquement pour montrer comment les paramètres sont spécifiés dans les appels emboités. Ils n'ont pas de signification logique.

Prenons le cas d'une macro LEVEL1, dont la définition est :

```
MACRO
$LAB = LEVEL1  $P1,$P2,$P3,$P4,$P5
      B        $P1
      A        $P2
$LAB = *LEVEL2 $P4,$P2,ZONE,SAVE
      SI       $P4+$P5
      C        $P3
MEND
```

La macro LEVEL2 est définie comme :

```
MACRO
$LAB =  LEVEL2  $ARG1,$ARG2,$ARG3,      X
              $ARG4
$LAB =  ML      $ARG3
      SH      ZL 23
      SB      $ARG4
      *LEVEL3 1,$ARG2
      A      $ARG1
MEND
```

La macro LEVEL3 est simplement :

```
MACRO
LEVEL3  $KAR1,$KAR2
SK      $KAR2+$KAR1
C      SAVE
MEND
```

Comme nous le constatons, lorsque LEVEL1 est appelée, l'appel de la macro LEVEL2 est automatiquement généré. Une fois appelée, LEVEL2 invoque LEVEL3. Par un seul appel nous avons généré trois autres, ce qui constitue le niveau d'emboîtement. Théoriquement ce niveau peut être infini; il est cependant limité par la taille des piles et des tables nécessaires pour contrôler ce mécanisme.

Le macro générateur limite ce niveau à 10, ce qui est plus que suffisant pour les applications pratiques.

Dans l'exemple précédent, si nous écrivons :

```
NAME = *LEVEL1 VAR1,VAR2,RES,BRANCH,2
```

Le groupe d'instructions suivant sera généré :

```
      B      VAR1
      A      VAR2
NAME = ML    ZONE
      SH     ZL 23
      SB     SAVE
      SK     VAR2+1
      C      SAVE
      A      BRANCH
      SI     BRANCH+2
      C      RES
```

3 - APPELS RECURSIFS

Lorsque, dans sa propre définition, une macro appelle une autre, elle génère, comme nous l'avons vu, un appel emboîté.

Cependant, au lieu d'appeler une autre, une macro peut s'appeler elle-même, ce qui constitue un appel récursif. Les appels récursifs sont analysés par le macro-générateur comme des appels emboîtés ordinaires. Etant donné ce fait, toutes les règles concernant les appels emboîtés et la transmission de paramètres s'applique également aux

appels récursifs. Certes, un test doit être fait lorsque l'on utilise la récursivité afin d'éviter une boucle sans fin. En effet, compte tenu des limitations pratiques, le macro-générateur détectera une erreur, si la macro s'appelle elle-même plus de 10 fois consécutives.

Pour illustrer le concept de la récursivité, considérons le cas d'une macro qui transfère des éléments d'une table à une autre.

Cela peut naturellement être fait en provoquant une boucle pendant le temps d'exécution, ou, en définissant une macro du type :

```
MACRO
TRANSF  $PAR1, $PAR2, $N
DCGA    COUNT
$$COUNT = SETA    $$COUNT+1
B        $PAR1+$$COUNT
C        $PAR2+$$COUNT
CAB      ($$COUNT GE $N).END
*TRANSF  $PAR1,$PAR2,$N
.END      = MEND
```

Si nous écrivons :

```
TRANSF  ZONE1, ZONE2, 3
```

le code généré sera :

```
B        ZONE1+1
C        ZONE2+1
B        ZONE1+2
C        ZONE2+2
B        ZONE1+3
C        ZONE2+3
```

Les macros récursives sont très puissantes dans le calcul de certains types de fonctions qui ne peuvent être définies que par les moyens de la récursivité.

ANNEXE 1

Exemples d'application

- 1) Le premier exemple, montre l'utilisation du macro-générateur pour mettre à jour une définition de macro système.

Il s'agit de la macro IFB (if bit) utilisée pour comparer des chaînes de bits.

Après l'analyse de sa définition, la macro est écrite sur disque sous une forme interne, qui remplace automatiquement la version précédente. Cette procédure est extrêmement commode pour une constante mise à jour de la bibliothèque système.

***** TAS-86 MACRO-GENERATOR ***

THE OPTIONS SPECIFIED FOR THE MACRO GENERATOR ARE

SOURCE
UPDATE
NODECK
NOLIST
NO-IDENT

```

1      MACRO
2      $LABEL = IFB    $STR1,$I1,$OPER,$STR2,$I2,$LENGTH,$LABO,$LABI
3                  --DECLARATION OF INTERNAL VARIABLES--
4                  --GLOBAL VARIABLES--
5      DCGA  WIFB,WBBI
6                  --LOCAL VARIABLE--
7      DCLC  TESTW
8                  --TEST IF WE WANT JUST TO TEST ONE STRING--
9      CAB   ($LENGTH NE ' ') .OK0
10         --IF YES,CALL MACRO TESTB,AND JUMP TO END--
11      $LABEL = *TESTB $STR1,$I1,$OPER,$STR2,$I2
12      UAB   .END
13         --FILLING OF SUPPLYING WORDS FOR THE SUBROUTINE--
14         --FILLING OF 1ST SUPPLYING WORD--
15      .OK0   = NOP
16      $LABEL = B      $STR1      ,--FILLING OF 1ST SUPPLYING WORD AND--
17                  C      WWWC1.   ,--STORAGE IN AN INTERNAL WORK VARIABLE--
18                  --FILLING OF 2ND SUPPLYING WORD--
19                  --TEST IF BIT ORDER IS A DOPE VECTOR--
20      $$TESTW= SETC    MOVE($I1,1,1,1)
21                  CAB   ($$TESTW EQ W) .OK1
22                  B      $I1      ,--FILLING OF 2ND SUPPLYING WORD AND--
23                  UAB   .OK2
24      .OK1   = BE      $I1      ,--FILLING OF 2ND SUPPLYING WORD AND--
25      .OK2   = C      WWWC2.   ,--STORAGE IN AN INTERNAL WORK VARIABLE--
26                  --FILLING OF 3RD SUPPLYING WORD--
27                  B      $STR2    ,--FILLING OF 3RD SUPPLYING WORD--
28                  C      WWWC3.   ,--STORAGE IN AN INTERNAL WORK VARIABLE--
29                  --FILLING OF 4TH SUPPLYING WORD--
30                  --TEST IF BIT ORDER IS A DOPE VECTOR--
31      $$TESTW= SETC    MOVE($I2,1,1,1)
32                  CAB   ($$TESTW EQ W) .OK3
33                  B      $I2      ,--FILLING OF 4TH SUPPLYING WORD AND--
34                  UAB   .OK4
35      .OK3   = BE      $I2      ,--FILLING OF 4TH SUPPLYING WORD AND--
36      .OK4   = C      WWWC4.   ,--STORAGE IN AN INTERNAL WORK VARIABLE--
37                  --FILLING OF 5TH SUPPLYING WORD--
38                  --TEST IF THE LENGTH IS A DOPE VECTOR--
39      $$TESTW= SETC    MOVE($LENGTH,1,1,1)
40                  CAB   ($$TESTW EQ W) .OK5
41                  B      $LENGTH   ,--FILLING OF 5TH SUPPLYING WORD AND--
42                  UAB   .OK6
43      .OK5   = BE      $LENGTH   ,--FILLING OF 5TH SUPPLYING WORD AND--
44      .OK6   = C      WWWC5.   ,--STORAGE IN AN INTERNAL WORK VARIABLE--
45                  --FILLING OF 6TH SUPPLYING WORD--
46                  --TEST TO DIFFERENCIATE THE OPERATORS--
47                  CAB   ($OPER NE NE) .OK8
48                  B      -1        ,--FILLING 6TH SUPPLYING WORD (OPER. NE)--
49                  UAB   .OK10
50      .OK8   = CAB   ($OPER NE EQ) .ERROR
51      .OK9   = SH     V      0    ,--FILLING 6TH SUPPLYING WORD (OPER. EQ)--
52                  UAB   .OK10
53      .ERRCR = NOTE    INVALID OPERATOR IN MACRO IFB,EQ IS ASSUMED
54                  UAB   .OK9
55      .OK10  = C      WWWC0.   ,--STORAGE IN AN INTERNAL WORK VARIABLE--
56                  SU      WWWIFB. ,--SUBROUTINE CALL FOR PROCESSING--
57                  --FILLING OF 7TH SUPPLYING WORD WITH ADD OF 1ST LAB.--
58                  S      $LABO    ,--CONDITION SATISFIED,JUMP TO 1ST LABEL--
59                  --TEST IF 2ND LABEL EXIST--

```

~~A14~~

```
60          CAB      ($LAB1 EQ ' ') .OK11
61          --IF YES,FILLING OF 8TH SUPPLYING WORD--
62          --WITH THE ADDRESS OF THE 2ND LABEL--
63          S        $LAB1      ,--COND. NOT SATISFIED,JUMP TO 2ND LABEL--
64      .OK11  = NOP
65          --INITIALIZATION FOR SUBROUTINE GENERATION--
66      $$WIFB = SETA  1
67      $$WBBI = SETA  1
68      .END      = MEND
THE MACRO IFB      ,HAS BEEN REPLACED IN THE LIBRARY
69      ND ,
```

ANNEXE I-2

EXEMPLE DE MULTIPLICATION DE MATRICES

- 2) Cet exemple montre l'utilisation du macro-générateur et du langage macro-coral pour calculer un produit de matrices. Le programme comprend aussi la lecture des données et la sortie des résultats. Les commentaires sur le listing expliquent d'une façon assez détaillée, la logique générale du programme.

~~A16~~

***** TAS-86 MACRO-GENERATOR *****

THE OPTIONS SPECIFIED FOR THE MACRO GENERATOR ARE--

SOURCE
GENERATE
NODECK
LIST
NO-IDENT

```

1  -- EXEMPLE DE MULTIPLICATION DES MATRICES--
2  --AYANT TROIS MATRICES TAB1,TAB2 ET TAB3,ON CALCULE TAB3=TAB1*TAB2--
3  ZRAICK = *BEGIN
4  -- DECLARATION DES TROIS MATRICES--
5      *DCL TAB1,WORD,10,10
6      *DCL TAB2,WORD,10,10
7      *DCL TAB3,WORD,10,10
8  --BOUCLE POUR LA LECTURE DES ELEMENTS DE TAB1 ET TAB2--
9      *FOR I,1,1,10
10     *FOR J,1,1,10
11 --ADRESSE DE TAB1(I,J)--
12     *ARRAY TAB1,I,J,WDOPI
13 --ADRESSE DE TAB2(I,J)--
14     *ARRAY TAB2,I,J,WDOPI
15 --LECTURE SUR UNE CARTE DE TAB1(I,J) ET DE TAB2(I,J)--
16 --LA PRECISION EST DE 6 CHIFFRES SEPARES PAR 4 BLANCS
17     *READ D,6,WDOPI,X,4,,D,6,WDOPI,S
18     *END
19     *END
20 --BOUCLE EXTERNE POUR REMPLIR LES LIGNES DE TAB3--
21     *FOR K,1,1,10
22 --BOUCLE POUR REMPLIR LES COLONNES DE TAB3--
23     *FOR M,1,1,10
24 --OBTENTION DE L'ADRESSE DE TAB3(K,M)--
25     *ARRAY TAB3,K,M,WDOPI
26 --INITIALISATION DE TAB3(K,M)--
27     *ASSIGN WDOPI,0
28 -- CALCUL DE TAB3(K,M)--
29     *FOR I,1,1,10
30 --OBTENTION DE L'ADRESSE DE TAB1(K,I)--
31     *ARRAY TAB1,K,I,WDOPI
32 -- OBTENTION DE L'ADRESSE DE TAB2(I,M)--
33     *ARRAY TAB2,I,M,WDOPI
34     *ASSIGN WDOPI,WDOPI,*,WDOPI,+,WDOPI
35 L1 = *END
36 LABEL = *END
37     *END
38 --SORTIE DES ELEMENTS DE TAB3--
39     *FOR I,1,1,10
40     *FOR J,1,1,10
41     *ARRAY TAB3,I,J,WDOPI
42 --IMPRESSION DE 10 ELEMENTS(PRECISION 8 CHIFFRES) PAR LIGNE--
43     *PRINT D,8,WDOPI,X,4
44     *END
45 --SAUT DE LIGNE--
46     *PRINT S
47     *END
48     *END
49     ND,

```

```

***** TAS-66 GENERATED PROGRAM *****
1  -- EXEMPLE DE MULTIPLICATION DES MATRICES--
2  --AYANT TROIS MATRICES TAB1,TAB2 ET TAB3,ON CALCULE TAB3=TAB1*TAB2--
3  --ZRAICK = *BEGIN
4      NM ZRAICK
5      SU    WWWINI.  --SUBROUTINE CALL FOR INITIALIZATION--
6  -- DECLARATION DES TROIS MATRICES--
7  --      *DCL  TAB1,WORD,10,10
8  TAB1      =      (102)
9      SH      V 0      --AND PREPARATION FOR FILLING--
10     AA      TAB1+2    --OF THE 1ST DOPE VECTOR WORD--
11     AT      '800000'H
12     C      TAB1      --STORAGE OF THIS WORD--
13     B      10      --FILLING 2ND DOPE VECTOR WORD--
14     SH      L 12
15     AT      10
16     C      TAB1+1    --STORAGE OF THIS WORD--
17  --      *DCL  TAB2,WORD,10,10
18  TAB2      =      (102)
19     SH      V 0      --AND PREPARATION FOR FILLING--
20     AA      TAB2+2    --OF THE 1ST DOPE VECTOR WORD--
21     AT      '800000'H
22     C      TAB2      --STORAGE OF THIS WORD--
23     B      10      --FILLING 2ND DOPE VECTOR WORD--
24     SH      L 12
25     AT      10
26     C      TAB2+1    --STORAGE OF THIS WORD--
27  --      *DCL  TAB3,WORD,10,10
28  TAB3      =      (102)
29     SH      V 0      --AND PREPARATION FOR FILLING--
30     AA      TAB3+2    --OF THE 1ST DOPE VECTOR WORD--
31     AT      '800000'H
32     C      TAB3      --STORAGE OF THIS WORD--
33     B      10      --FILLING 2ND DOPE VECTOR WORD--
34     SH      L 12
35     AT      10
36     C      TAB3+1    --STORAGE OF THIS WORD--
37  --BOUCLE POUR LA LECTURE DES ELEMENTS DE TAB1 ET TAB2--
38  --      *FOR  I,1,1,10
39     B      1      --BRING INITIAL VALUE--
40     S      3L
41  WWWB1     = B      I      --BRING LOOP VARIABLE--
42     A      1      --ADD THE STEP--
43     C      I      --TO HAVE THE NIEW VALUE OF THE LOOP VAR--
44     SB     10      --TEST FOR END OF THE LOOP--
45     SK     3L      --IF NO,EXECUTE ONE MORE TIME THE--
46     SI     2L      --INSTRUCTIONS IN THE LOOP--
47     S      WWWE1+1 --IF YES,RETURN JUMP OR END JUMP--
48  --      *FOR  J,1,1,10
49     B      1      --BRING INITIAL VALUE--
50     S      3L
51  WWWB2     = B      J      --BRING LOOP VARIABLE--
52     A      1      --ADD THE STEP--
53     C      J      --TO HAVE THE NIEW VALUE OF THE LOOP VAR--
54     SB     10      --TEST FOR END OF THE LOOP--
55     SK     3L      --IF NO,EXECUTE ONE MORE TIME THE--
56     SI     2L      --INSTRUCTIONS IN THE LOOP--
57     S      WWWE2+1 --IF YES,RETURN JUMP OR END JUMP--
58  --ADRESSE DE TAB1(I,J)--

```

```

59  --      AB9
60      *ARRAY TAB1,I,J,WDOPI
61      B      TAB1      --FILLING OF THE DOPE VECTOR --
62      ET      '7FFFFFF'H --(FIRST PART) WITHOUT THE--
63      C      WDOPI      --ARRAY SPECIFICATION CODE--
64      B      TAB1+1     --BRINGING OF THE SECOND DOPE VECTOR--
65      C      WWDOP1     --STORAGE OF THIS WORD--
66      ET      'FFF'H    --EXTRACTION OF THE SECOND DIMENSION--
67      C      WWWORK.    --STORAGE OF THIS DIMENSION--
68      ML      I         --WORD NUMBER BEFORE THE FIRST ITH ELEM.--
69      SH      LZ 23     --RIGHT JUSTIFICATION IN ACCUMULATOR--
70      SB      WWWORK.   --RELATIVE ADDRESS CALCULATION--
71      A      J         --RELATIVE ADDRESS CALCULATION (+1) --
72      SA      IU       --EXACT RELATIVE ADDRESS OF THE ELEMENT--
73      A      WDOPI     --ABSOLUTE ADDRESS OF THE REQUIRED ELEM.--
74      C      WDOPI     --LOADING OF THE REQUIRED DOPE VECTOR--
75  --ADRESSE DE TAB2(I,J)--
76  --      *ARRAY TAB2,I,J,WDOPI
77      B      TAB2      --FILLING OF THE DOPE VECTOR --
78      ET      '7FFFFFF'H --(FIRST PART) WITHOUT THE--
79      C      WDOPI     --ARRAY SPECIFICATION CODE--
80      B      TAB2+1     --BRINGING OF THE SECOND DOPE VECTOR--
81      C      WWDOP2     --STORAGE OF THIS WORD--
82      ET      'FFF'H    --EXTRACTION OF THE SECOND DIMENSION--
83      C      WWWORK.    --STORAGE OF THIS DIMENSION--
84      ML      I         --WORD NUMBER BEFORE THE FIRST ITH ELEM.--
85      SH      LZ 23     --RIGHT JUSTIFICATION IN ACCUMULATOR--
86      SB      WWWORK.   --RELATIVE ADDRESS CALCULATION--
87      A      J         --RELATIVE ADDRESS CALCULATION (+1) --
88      SA      IU       --EXACT RELATIVE ADDRESS OF THE ELEMENT--
89      A      WDOPI     --ABSOLUTE ADDRESS OF THE REQUIRED ELEM.--
90      C      WDOPI     --LOADING OF THE REQUIRED DOPE VECTOR--
91  --LECTURE SUR UNE CARTE DE TAB1(I,J) ET DE TAB2(I,J)--
92  --LA PRECISION EST DE 6 CHIFFRES SEPARÉS PAR 4 BLANCS
93  --      *READ D,6,WDOPI,X,4,,D,6,WDOPI,S
94      SUEY    GENORM
95      '1105'H
96      AA      ERROR1.
97      SUEY    EINVOR
98      '5'H
99      S      -2L
100     B      WDOPI
101     ET      'FFFF'H
102     C      WWWA1.
103     B      6
104     SH      L 15
105     AT      '2805C0'H
106     C      2L
107     SUEY    EINKVH
108     0
109     S      ERROR3.
110     CE      WWWA1.
111     B      1          --BRING INITIAL VALUE--
112     S      3L
113     WWWB3 = B      WWWI --BRING LOOP VARIABLE--
114     A      1          --ADD THE STEP--
115     C      WWWI      --TO HAVE THE NEW VALUE OF THE LOOP VAR--
116     SB      4         --TEST FOR END OF THE LOOP--
117     SK      3L        --IF NO,EXECUTE ONE MORE TIME THE--
118     SI      2L        --INSTRUCTIONS IN THE LOOP--
119     S      WWWE3+1    --IF YES,RETURN JUMP OR END JUMP--

```

```

119      SUEY  EINGAB
120  WWWE3  =  S    WWWB3      --RETURN JUMP--
121      B      WDOP2
122      ET      'FFFF'H
123      C      WWWA1.
124      H      6
125      SH      L 15
126      AT      '2805C0'H
127      C      2L
128      SUEY  EINKVH
129      O
130      S      ERROR3.
131      CE      WWWA1.
132      SUEY  BRIZET
133      '227'H
134      S      -2L
135  --      *END
136  WWWE2  =  S    WWWB2      --RETURN JUMP--
137  --      *END
138  WWWE1  =  S    WWWB1      --RETURN JUMP--
139  --BOUCLE EXTERNE POUR REMPLIR LES LIGNES DE TAB3--
140  --      *FOR  K,1,1,10
141      B      1      --BRING INITIAL VALUE--
142      S      3L
143  WWWB4  =  B      K      --BRING LOOP VARIABLE--
144      A      1      --ADD THE STEP--
145      C      K      --TO HAVE THE NIEW VALUE OF THE LOOP VAR--
146      SB     10      --TEST FOR END OF THE LOOP--
147      SK     3L      --IF NO,EXECUTE ONE MORE TIME THE--
148      SI     2L      --INSTRUCTIONS IN THE LOOP--
149      S      WWWE4+1  --IF YES,RETURN JUMP OR END JUMP--
150  --BOUCLE POUR REMPLIR LES COLONNES DE TAB3--
151  --      *FOR  M,1,1,10
152      B      1      --BRING INITIAL VALUE--
153      S      3L
154  WWWB5  =  B      M      --BRING LOOP VARIABLE--
155      A      1      --ADD THE STEP--
156      C      M      --TO HAVE THE NIEW VALUE OF THE LOOP VAR--
157      SB     10      --TEST FOR END OF THE LOOP--
158      SK     3L      --IF NO,EXECUTE ONE MORE TIME THE--
159      SI     2L      --INSTRUCTIONS IN THE LOOP--
160      S      WWWE5+1  --IF YES,RETURN JUMP OR END JUMP--
161  --OBTENTION DE L'ADRESSE DE TAB3(K,M)--
162  --      *ARRAY TAB3,K,M,WDOP3
163      B      TAB3      --FILLING OF THE DOPE VECTOR --
164      ET      '7FFFFFFF'H --(FIRST PART) WITHOUT THE--
165      C      WDOP3      --ARRAY SPECIFICATION CODE--
166      B      TAB3+1      --BRINGING OF THE SECOND DOPE VECTOR--
167      C      WWDOP3      --STORAGE OF THIS WORD--
168      ET      'FFF'H      --EXTRACTION OF THE SECOND DIMENSION--
169      C      WWWORK.      --STORAGE OF THIS DIMENSION--
170      ML      K      --WORD NUMBER BEFORE THE FIRST ITH ELEM.--
171      SH      LZ 23      --RIGHT JUSTIFICATION IN ACCUMULATOR--
172      SB      WWWORK.      --RELATIVE ADDRESS CALCULATION--
173      A      M      --RELATIVE ADDRESS CALCULATION (+1) --
174      SA      1U      --EXACT RELATIVE ADDRESS OF THE ELEMENT--
175      A      WDOP3      --ABSOLUTE ADDRESS OF THE REQUIRED ELEM.--
176      C      WDOP3      --LOADING OF THE REQUIRED DOPE VECTOR--
177  --INITIALISATION DE TAB3(K,M)--
178  --      *ASSIGN WDOP3,0

```

~~AZ11~~

```
179      B      0      --BRING 1ST VARIABLE--
180      CE      WDO P3  --STORAGE OF THE RESULT--
181  -- CALCUL DE TAB3(K,M)--
182  --      *FOR I,1,1,10
183      B      1      --BRING INITIAL VALUE--
184      S      3L
185  WWB6 = B      I      --BRING LOOP VARIABLE--
186      A      1      --ADD THE STEP--
187      C      I      --TO HAVE THE NEW VALUE OF THE LOOP VAR--
188      SB     10      --TEST FOR END OF THE LOOP--
189      SK     3L      --IF NO,EXECUTE ONE MORE TIME THE--
190      SI     2L      --INSTRUCTIONS IN THE LOOP--
191      S      WWB6+1  --IF YES,RETURN JUMP OR END JUMP--
192  --OBTENTION DE L'ADRESSE DE TAB1(K,I)--
193  --      *ARRAY TAB1,K,I,WDO P1
194      B      TAB1    --FILLING OF THE DOPE VECTOR --
195      ET     '7FFFFFF'H --(FIRST PART) WITHOUT THE--
196      C      WDO P1  --ARRAY SPECIFICATION CODE--
197      B      TAB1+1  --BRINGING OF THE SECOND DOPE VECTOR--
198      C      WWDO P1 --STORAGE OF THIS WORD--
199      ET     'FFF'H   --EXTRACTION OF THE SECOND DIMENSION--
200      C      WWWORK.  --STORAGE OF THIS DIMENSION--
201      ML     K        --WORD NUMBER BEFORE THE FIRST ITH ELEM.--
202      SH     LZ 23    --RIGHT JUSTIFICATION IN ACCUMULATOR--
203      SB     WWWORK.  --RELATIVE ADDRESS CALCULATION--
204      A      I        --RELATIVE ADDRESS CALCULATION (+1) --
205      SA     10       --EXACT RELATIVE ADDRESS OF THE ELEMENT--
206      A      WDO P1   --ABSOLUTE ADDRESS OF THE REQUIRED ELEM.--
207      C      WDO P1   --LOADING OF THE REQUIRED DOPE VECTOR--
208  -- OBTENTION DE L'ADRESSE DE TAB2(I,M)--
209  --      *ARRAY TAB2,I,M,WDO P2
210      B      TAB2    --FILLING OF THE DOPE VECTOR --
211      ET     '7FFFFFF'H --(FIRST PART) WITHOUT THE--
212      C      WDO P2  --ARRAY SPECIFICATION CODE--
213      B      TAB2+1  --BRINGING OF THE SECOND DOPE VECTOR--
214      C      WWDO P2 --STORAGE OF THIS WORD--
215      ET     'FFF'H   --EXTRACTION OF THE SECOND DIMENSION--
216      C      WWWORK.  --STORAGE OF THIS DIMENSION--
217      ML     I        --WORD NUMBER BEFORE THE FIRST ITH ELEM.--
218      SH     LZ 23    --RIGHT JUSTIFICATION IN ACCUMULATOR--
219      SB     WWWORK.  --RELATIVE ADDRESS CALCULATION--
220      A      M        --RELATIVE ADDRESS CALCULATION (+1) --
221      SA     10       --EXACT RELATIVE ADDRESS OF THE ELEMENT--
222      A      WDO P2   --ABSOLUTE ADDRESS OF THE REQUIRED ELEM.--
223      C      WDO P2   --LOADING OF THE REQUIRED DOPE VECTOR--
224  --      *ASSIGN WDO P3,WDO P1,*,WDO P2,+,WDO P3
225      BE     WDO P1   --BRING 1ST VARIABLE--
226      MLE    WDO P2   --MULTIPLICATION BY THE VARIABLE--
227      SH     LZ 23    --(RESULT IN ACCUMULATOR)--
228      AE     WDO P3   --ADDITION OF THE VARIABLE--
229      CE     WDO P3   --STORAGE OF THE RESULT--
230  --L1 = *END
231  WWB6 = L1=S WWB6    --RETURN JUMP--
232  --LABEL = *END
233  WWB5 = LABEL=S WWB5 --RETURN JUMP--
234  --      *END
235  WWB4 = S WWB4       --RETURN JUMP--
236  --SORTIE DES ELEMENTS DE TAB3--
237  --      *FOR I,1,1,10
238      B      1      --BRING INITIAL VALUE--
```

-A12-

239	S	3L				+
240	WWB7 =	I	--BRING LOOP VARIABLE--			+
241	A	1	--ADD THE STEP--			+
242	C	1	--TO HAVE THE NEW VALUE OF THE LOOP VAR--			+
243	SH	10	--TEST FOR END OF THE LOOP--			+
244	SK	3L	--IF NO,EXECUTE ONE MORE TIME THE--			+
245	SI	2L	--INSTRUCTIONS IN THE LOOP--			+
246	S	WWB7+1	--IF YES,RETURN JUMP OR END JUMP--			+
247	--	*FOR J,1,1,10				
248	B	1	--BRING INITIAL VALUE--			+
249	S	3L				+
250	WWB8 =	J	--BRING LOOP VARIABLE--			+
251	A	1	--ADD THE STEP--			+
252	C	J	--TO HAVE THE NEW VALUE OF THE LOOP VAR--			+
253	SH	10	--TEST FOR END OF THE LOOP--			+
254	SK	3L	--IF NO,EXECUTE ONE MORE TIME THE--			+
255	SI	2L	--INSTRUCTIONS IN THE LOOP--			+
256	S	WWB8+1	--IF YES,RETURN JUMP OR END JUMP--			+
257	--	*ARRAY TAB3,I,J,WDOP3				
258	B	TAB3	--FILLING OF THE DOPE VECTOR --			+
259	ET	'7FFFFF'H	--(FIRST PART) WITHOUT THE--			+
260	C	WDOP3	--ARRAY SPECIFICATION CODE--			+
261	B	TAB3+1	--BRINGING OF THE SECOND DOPE VECTOR--			+
262	C	WDOP3	--STORAGE OF THIS WORD--			+
263	ET	'FFF'H	--EXTRACTION OF THE SECOND DIMENSION--			+
264	C	WWWORk.	--STORAGE OF THIS DIMENSION--			+
265	ML	I	--WORD NUMBER BEFORE THE FIRST ITH ELEM.--			+
266	SH	LZ 23	--RIGHT JUSTIFICATION IN ACCUMULATOR--			+
267	SB	WWWORk.	--RELATIVE ADDRESS CALCULATION--			+
268	A	J	--RELATIVE ADDRESS CALCULATION (+1) --			+
269	SA	1U	--EXACT RELATIVE ADDRESS OF THE ELEMENT--			+
270	A	WDOP3	--ABSOLUTE ADDRESS OF THE REQUIRED ELEM.--			+
271	C	WDOP3	--LOADING OF THE REQUIRED DOPE VECTOR--			+
272	--	IMPRESSION DE 10 ELEMENTS(PRECISION 8 CHIFFRES) PAR LIGNE--				
273	--	*PRINT D,8,WDOP3,X,4				
274	SUEY	GENORM				+
275		'4'H				+
276	AA	ERROR2.				+
277	SUEY	AUSVOR				+
278		'4'H				+
279	S	-2L				+
280	S	-3L				+
281	B	WDOP3				+
282	ET	'FFFF'H				+
283	C	WWA1.				+
284	B	8				+
285	SH	L 3				+
286	AT	'3270000'D				+
287	C	3L				+
288	BE	WWA1.				+
289	SUEY	AUSKVZ				+
290		0				+
291	B	1	--BRING INITIAL VALUE--			+
292	S	3L				+
293	WWB9 =	WWI	--BRING LOOP VARIABLE--			+
294	A	1	--ADD THE STEP--			+
295	C	WWI	--TO HAVE THE NEW VALUE OF THE LOOP VAR--			+
296	SH	4	--TEST FOR END OF THE LOOP--			+
297	SK	3L	--IF NO,EXECUTE ONE MORE TIME THE--			+
298	SI	2L	--INSTRUCTIONS IN THE LOOP--			+

~~AT13~~

```
299          S      WWWE9+1  --IF YES, RETURN JUMP OR END JUMP--      +
300          B      42                                     +
301          SUEY  AUSGAB                                     +
302  WWWE9  =  S      WWWB9  --RETURN JUMP--                          +
303  --          *END                                     +
304  WWWE8  =  S      WWWB8  --RETURN JUMP--                          +
305  --SAUT DE LIGNE--                                     +
306  --          *PRINT S                                     +
307          B      39                                     +
308          SUEY  AUSGAB                                     +
309  --          *END                                     +
310  WWWE7  =  S      WWWB7  --RETURN JUMP--                          +
311  --          *END                                     +
312          SUEY  START                                     +
313  WWWINI.=  0      --SUBROUTINE FOR INITIALIZATION, IF WANTED--    +
314          SE      WWWINI.  --INITIALIZATION NOT WANTED, RETURN JUMP-- +
315          ND,
```

315 TAS-86 INSTRUCTIONS HAVE BEEN GENERATED

ANNEXE 2

Les pages suivantes contiennent une description
détaillée des macros systèmes qui constituent le
langage MACRO CORAL, ainsi que leur définition.

***** DEFINITION DU LANGUAGE 'MACRO-CORAL' *****

LES PAGES SUIVANTES, CONTIENNENT LES DEFINITIONS DES MACROS INSTRUCTIONS, QUI SONT LES ELEMENTS DE BASE DU LANGUAGE 'MACRO-CORAL'

CES DEFINITIONS, NE SONT QU'UNE PREMIERE VERSION DE CE LANGUAGE. ACTUELLEMENT ON TRAVAILLE SUR CES DEFINITIONS, AFIN DE MINIMISER LEUR FACTEUR D'EXPANSION, AINSI QUE LEUR PERFORMANCE AU POINT DE VUE TEMPS D'EXECUTION.

```

MACRO
$PRONAM= BEGIN $PAR1
    --DECLARATION OF INTERNAL VARIABLES--
    --GLOBAL VARIABLE--
    DCGA WCHECK
    --LOCAL VARIABLE--
    DCLC NMCARD
    --TEST FOR CHECKING--
    CAB ($PAR1 NE CHECK) .OK1
    --IF CHECKING IS WANTED, SETTING OF A SWITCH--
$$WCHECK=SETA 1
    --OUTPUT OF THE PROGRAM NAME IN TAS 86 FORM--
.OK1 = NOP
$$NMCARD=SETC NM
$$NMCARD=SETC MOVE($PRONAM,1,6,4)
    $NMCARD
    SU WWWINI. --SUBROUTINE CALL FOR INITIALIZATION--
MEND

```

```

MACRO
DCL $NAME,$TYPE,$DIM1,$DIM2,$INIT,$ALIAS
    --DECLARATION OF INTERNAL VARIABLES--
    --LOCAL VARIABLES--
    DCLA DIM,DV1,DV11,LNDV
    DCLC DV21,DV22
    --TEST FOR ARITHMETIC ARRAY CASE--
    -- OR FOR STRING CASES --
    CAB ($TYPE NE WORD) .STRING
    --ARITHMETIC ARRAY PROCESSING--
$$DIM = SETA $DIM1
$$DV21 = SETC $DIM1
    --TEST FOR TWO DIMENSIONAL ARRAY--
    CAB ($DIM2 EQ ' ') .ONEDIM
    --WORD NUMBER CALCULATION IF TWO DIMENSIONS--
$$DIM = SETA $$DIM*$DIM2
    --SETTING OF TWO DIMENSIONAL ARRAY SPECIFICATION--
$$DV1 = SETA 800000
$$DV22 = SETC $DIM2
.ONEDIM= NOP
    --ARRAY LENGTH CALCULATION--
$$DIM = SETA $$DIM+2
    --ARRAY RESERVATION (IF WANTED)--
    CAB ($INIT EQ INIT) .NO1
$NAME = ($$DIM)
    SH V 0 --AND PREPARATION FOR FILLING--
    AA $NAME+2 --OF THE 1ST DOPE VECTOR WORD--
    UAB .C1
    --FILLING OF THE DOPE VECTOR WORDS--
.NO1 = NOP
$NAME = (2)
    SH V 0 --RESERVATION FOR THE DOPE VECTOR--
    AA $ALIAS --FILLING OF 1ST DOPE VECTOR WORD--
.C1 = AT '$$DV1'H
    C $NAME --STORAGE OF THIS WORD--
    B $$DV21 --FILLING 2ND DOPE VECTOR WORD--
    --TEST FOR TWO DIMENSIONAL ARRAY--
    CAB ($DIM2 NE ' ') .TWO DIM
$$DV22 = SETC 1
.TWO DIM= SH L 12
    AT $$DV22
    C $NAME+1 --STORAGE OF THIS WORD--
    UAB .END

```

```

--STRING PROCESSING--
.STRING= NOP
$$DV21 = SETC $DIM1 PD/TM.
--TEST FOR 6 BIT CHARACTER STRINGS-- PD/TM.
CAB ($TYPE EQ CHAR) .CHAR6B PD/TM.
--TEST FOR BIT STRINGS--
CAB ($TYPE EQ BIT) .BITSTR PD/TM.
--TEST FOR 8 BIT CHARACTER STRINGS--
CAB ($TYPE EQ CHAR8) .CHAR8B PD/TM.
--IF NO, ERROR MESSAGE-- PD/TM.
NOTE ILLEGAL TYPE SPECIFICATION, ERROR IN 2ND PARAMETER PD/TM.
UAB .END PD/TM.
--CALCULATION OF WORD NUMBER PER STRING--
--FOR 6 BIT CHARACTER STRINGS--

.CHAR6B= NOP PD/TM.
$$DIM = SETA $DIM1+3 PD/TM.
$$DIM = SETA $$DIM/4 PD/TM.
UAB .RESERV PD/TM.
--FOR BIT STRINGS--

.BITSTR= NOP PD/TM.
$$DIM = SETA $DIM1+23 PD/TM.
$$DIM = SETA $$DIM/24 PD/TM.
UAB .RESERV PD/TM.
--FOR 8 BIT CHARACTER STRINGS--

.CHAR8B= NOP PD/TM.
$$DIM = SETA $DIM1+2 PD/TM.
$$DIM = SETA $$DIM/3 PD/TM.
--CALCULATION OF WORD NUMBER TO RESERVE--

.RESERV= NOP PD/TM.
$$DV22 = SETC $$DIM PD/TM.
--TEST FOR STRING ARRAYS--
CAB ($DIM2 NE ' ') .ARRAY PD/TM.
UAB .WORNUM PD/TM.
--CALCULATION OF WORD NUMBER OF THE ARRAY--

.ARRAY = NOP PD/TM.
$$DIM = SETA $$DIM*$$DIM2 PD/TM.
--SETTING OF THE ARRAY SPECIFICATION--
$$DV11 = SETA 800000 PD/TM.
--CALCULATION OF THE TOTAL WORD NUMBER--

.WORNUM= NOP PD/TM.
$$DIM = SETA $$DIM+2 PD/TM.
--RESERVATION OF THE FIELD (IF WANTED)--
CAB ($INIT EQ INIT) .NO2
$NAME = ($$DIM)
UAB .NO3
.NO2 = NOP
$NAME = (2) --RESERVATION FOR THE DOPE VECTOR--
--TEST FOR TYPE SPECIFICATION SETTING--
.NO3 = CAB ($TYPE EQ CHAR) .POSTY1 PD/TM2
CAB ($TYPE EQ BIT) .POSTY2
--SETTING OF THE TYPE SPECIFICATION--
--FOR 8 BIT CHARACTER STRINGS--
$$DV1 = SETA 30000 PD/TM2
UAB .FILLIN PD/TM2
--FOR 6 BIT CHARACTER STRINGS--
.POSTY1= NOP PD/TM2
$$DV1 = SETA 20000 PD/TM2
UAB .FILLIN PD/TM2
--FOR BIT STRINGS--
.POSTY2= NOP PD/TM2
$$DV1 = SETA 40000 PD/TM2
--FILLING OF THE DOPE VECTOR WORDS--
.FILLIN= B '$$DV11'H --FILLING OF 1ST DOPE VECTOR WORD--
AT '$$DV1'H
CAB ($ALIAS NE ' ') .ALIAS
AA $NAME+2
UAB .C2
.ALIAS = AA $ALIAS
.C2 = C $NAME --STORAGE OF THIS WORD--
B $$DV21 --FILLING OF THE 2ND DOPE VECTOR WORD-- PD/TM.
SH L 12 PD/TM.
A $$DV22 PD/TM.
C $NAME+1 --STORAGE OF THIS WORD-- PD/TM.
.END = MEND

```

```

MACRO
DECLTAB $NAME,$ITEMS,$ENTRYS,$ENTRWS,$INIT,$ALIAS
--DECLARATION OF INTERNAL VARIABLES--

```

```

--GLOBAL VARIABLES--
DCGA WDPNT,WDVZON
DCGC WTABNM,WALIAS
--LOCAL VARIABLE--
DCLA DIM
DCLC TESTW
--CALCULATION OF WORD NUMBER TO RESERVE--
--ITEMS DOPE VECTOR WORD NUMBER--
$$WDVZON=SETA $ITEMS*3
--TOTAL DOPE VECTOR WORD NUMBER--
$$WDVZON=SETA $$WDVZON+2
--TABLE WORD NUMBER--
$$DIM =SETA $ENTRYS*$ENTRWS
--TOTAL WORD NUMBER TO RESERVE--
$$DIM =SETA $$DIM+$$WDVZON
SH V O
--TABLE RESERVATION (IF WANTED)--
CAB ($INIT EQ INIT) .NO1
$NAME =($$DIM)
AA $NAME+$$WDVZON
UAB .FILLIN
.NO1 = NOP
$NAME = ($$WDVZON) --RESERVATION FOR THE DOPE VECTOR--
--FILLING OF THE DOPE VECTOR WORDS--
AA $ALIAS
$$TESTW= SETC MOVE($ALIAS,1,1,1)
CAB ($$TESTW NE W) .FILLIN
AT '020000'H
.FILLIN= C $NAME
B $ITEMS
SH LK 12
AT $ENTRWS
C $NAME+1 --STORAGE OF THIS WORD--
--INITIALIZATION OF THE GLOBAL VARIABLES--
--FOR THE MACRO ITEM --
$$WDVZON=SETA $$WDVZON-1
--FOR THE MACROS ITEM AND ENDTAB --
$$WDVPNT=SETA 1
$$WTABNM=SETC $NAME
MEND

MACRO
ITEM $NAME,$TYPE,$LENGTH,$OFSTWO,$OFSTBI
--DECLARATION OF INTERNAL VARIABLES--
--GLOBAL VARIABLES--
DCGA WDPNT,WDVZON
DCGC WTABNM,WALIAS
--LOCAL VARIABLES--
DCLA LENG$,$BITINI,$DISPLA
DCLC ITNAME,IT1,IT2
--CALCULATION OF RELATIVE ADDRESS OF DOPE VECTOR--
$$WDVPNT=SETA $$WDVPNT+1
--EXTENSION OF ITEM NAME TO 8 CHARACTERS--
$$ITNAME=SETC $NAME
$$ITNAME=SETC MOVE(00,1,2,7)
$$IT1 =SETC MOVE('',1,2,1)
$$IT1 = SETC MOVE($$ITNAME,1,4,3)
$$IT1 = SETC MOVE('',1,2,7)
$$IT2 = SETC MOVE($$ITNAME,5,4,1)
--FILLING OF THE DOPE VECTOR WORDS (3 PER ITEM)--
B $$IT1
C $$WTABNM+$$WDVPNT --STORAGE OF THIS WORD--
--CALCULATION OF RELATIVE ADDRESS OF DOPE VECTOR--
$$WDVPNT=SETA $$WDVPNT+1
B '$$IT2'
--TEST FOR WORD LENGTH PER STRING OR PER ARRAY--
CAB ($LENGTH EQ ' ') .LENGS1
--LENGTH CASE,SETTING OF THIS LENGTH--
$$LENGS =SETA $LENGTH
UAB .LENGS2
--NO LENGTH,SETTING OF 1 IN THIS CASE--
.LENGS1 =NOP
$$LENGS =SETA 1
.LENGS2 =AT $$LENGS --FILLING OF 2ND ITEM DOPE VECTOR WORD--
C $$WTABNM+$$WDVPNT --STORAGE OF THIS WORD--
--CALCULATION OF RELATIVE ADDRESS OF DOPE VECTOR--
$$WDVPNT=SETA $$WDVPNT+1
--TEST FOR BIT OFFSET--

```

```

CAB ($OFSTBI EQ ' ') .FIL3WO
--IF NO,INITIALIZATION OF AN INTERNAL VARIABLE--
$$BITINI=SETA $OFSTBI-1
.FIL3WO =B $BITINI --FILLING OF 3RD ITEM DOPE VECTOR WORD--
SH L 16
--TEST FOR ARITHMETIC ARRAYS--
CAB ($TYPE EQ WORD) .FILEND
--TEST FOR 6 BIT CHARACTER STRINGS--
CAB ($TYPE NE CHAR) .CHAR8B
AT '200000'H --FOR A 6 BIT CHARACTER STRING--
UAB .FILEND
--TEST FOR 8 BIT CHARACTER STRINGS--
.CHAR8B =CAB ($TYPE NE CHAR8) .BITSTR
AT '600000'H --FOR A 8 BIT CHARACTER STRING--
UAB .FILEND
--TEST FOR BIT STRINGS--
.BITSTR =CAB ($TYPE NE BIT) .ERROR
AT '400000'H --FOR A BIT STRING--
UAB .FILEND
--IF NO,ERROR MESSAGE--
.ERROR =NOTE ILLEGAL TYPE SPECIFICATION,ERROR IN 2ND PARAMETER
--CALCULATION OF WORD OFFSET (DISPLACEMENT)--
.FILEND =NOP
$$DISPLA=SETA $OFSTWO - 1
--CALCULATION OF BEGIN ADDRESS OF THE ITEM--
AA $$DISPLA&U
C $$WTABNM+$$WDVPNT
MEND

```

```

MACRO
ENDTAB $NAME
--DECLARATION OF INTERNAL VARIABLES--
--GLOBAL VARIABLES--
DCGA WDPNT
DCGC WTABNM
--TEST FOR NAME EQUALITY--
CAB ($$WTABNM EQ $NAME) .OK1
--IF NO,ERROR MESSAGE--
NOTE THE NAME SPECIFIED IN ENDTAB DOESN'T MATCH THE
NOTE NAME OF THE MOST RECENT NAME DECLARED IN DCLTAB.
NOTE THE LAST ONE IS ASSUMED.
--CALCULATION OF ITEM NUMBER--
.OK1 =NOP
$$WDVPNT=SETA $$WDVPNT/3
--TEST FOR ITEM NUMBERS EQUALITY--
B $$WTABNM
ET 'FF0000'H --ITEM NUMBER EXTRACTION--
SH KL 8
SB $$WDVPNT --TEST IF ITEM NUMBERS CORRESPOND--
S 2L
--IF ITEM NUMBERS DIFFER,ERROR MESSAGE--
SU WWWW4. --JUMP TO AN ERROR SUBROUTINE--
--INITIALIZATION OF A GLOBAL VARIABLE TO BLANKS--
$$WTABNM=SETC ' '
MEND

```

```

MACRO
$LABEL = ARRAY $NAME,$I,$J,$WDOPEV
--DECLARATION OF INTERNAL VARIABLES--
--GLOBAL VARIABLE--
DCGA WCHECK
--LOCAL VARIABLES--
DCLC TESTW,LABEL
--INITIALIZATION FOR A RIGHT LABEL GENERATION--
$$LABEL= SETC $LABEL
--TEST IF A CHECK IS WANTED--
CAB ($$WCHECK NE 1) .PROCES
$LABEL = B $NAME --CHECKING PART,BRINGING DOPE VECTOR--
ET 'FF0000'H --KEEPING OF THE FIRST 8 BITS--
--TEST FOR 2 DIMENSIONAL ARITHMETIC ARRAYS--
CAB ($J EQ ' ') .ONEDIM
--2 DIMENSIONAL ARITHMETIC ARRAY CASES--
AT '800000'H --TEST IF ARRAY SPECIFICATION CODE--
SI 2L --IS SET IN THE DOPE VECTOR WORD--
SU WWWW1. --IF NO,ERROR MESSAGE--
UAB .PROCES

```

--1 DIMENSIONAL ARITHMETIC ARRAY CASES--
 --OR DATA STRING ARRAY CASES--

```
.ONEDIM= SI 9L --TEST FOR 1 DIM. ARITHMETIC ARRAY--
        SK 2L --TEST FOR STRING ARRAY CASES--
        S 6L --IF NO (JUST A STRING), ERROR --
        ET '600000'H --TEST FOR TYPE SPECIFICATION CODE--
        SI 4L --IF ARITHMETIC TYPE, ERROR MESSAGE--
        B $NAME
        ET '1F0000'H --TEST FOR BIT OFFSET (MUST BE 0) --
        SI 2L
        SU WWW2. --IF NO, ERROR MESSAGE--
        --UPDATING OF THE LABEL FOR A RIGHT GENERATION--
$$LABEL= SETC ' '
        --PROCESSING PART OF THE MACRO--
.PROCES= NOP
$$LABEL= B $NAME --FILLING OF THE DOPE VECTOR --
        ET '7FFFFFFF'H --(FIRST PART) WITHOUT THE--
        C $WDOPEV --ARRAY SPECIFICATION CODE--
        --TEST IF THE NAME IS A DOPE VECTOR--
$$TESTW= SETC MOVE($NAME,1,1,1)
        CAB ($$TESTW NE W) .OK1
        B W&$NAME --BRINGING OF THE SECOND DOPE VECTOR--
        UAB .OK2
.OK1 = B $NAME+1 --BRINGING OF THE SECOND DOPE VECTOR--
.OK2 = C W&$WDOPEV --STORAGE OF THIS WORD--
        ET 'FFF'H --EXTRACTION OF THE SECOND DIMENSION--
        C WWWORK. --STORAGE OF THIS DIMENSION--
        --TEST IF 1ST DIMENSION IS A DOPE VECTOR--
$$TESTW= SETC MOVE($I,1,1,1)
        CAB ($$TESTW NE W) .OK3
        MLE $I --WORD NUMBER BEFORE THE FIRST ITH ELEM.--
        UAB .OK4
.OK3 = ML $I --WORD NUMBER BEFORE THE FIRST ITH ELEM.--
.OK4 = SH LZ 23 --RIGHT JUSTIFICATION IN ACCUMULATOR--
        SB WWWORK. --RELATIVE ADDRESS CALCULATION--
        --TEST FOR SECOND DIMENSION EXISTENCE--
        CAB ($J EQ ' ') .OK7
        --TEST IF 2ND DIMENSION IS A DOPE VECTOR--
$$TESTW= SETC MOVE($J,1,1,1)
        CAB ($$TESTW NE W) .OK5
        AE $J --RELATIVE ADDRESS CALCULATION (+1) --
        UAB .OK6
.OK5 = A $J --RELATIVE ADDRESS CALCULATION (+1) --
.OK6 = SA 1U --EXACT RELATIVE ADDRESS OF THE ELEMENT--
.OK7 = A $WDOPEV --ABSOLUTE ADDRESS OF THE REQUIRED ELEM.--
        C $WDOPEV --LOADING OF THE REQUIRED DOPE VECTOR--
        MEND
```

MACRO

```
$LABEL = TABLE $TABNAM,$IENTR,$ITNAME,$IELEM,$WDOPEV
        --DECLARATION OF INTERNAL VARIABLES--
        --GLOBAL VARIABLE--
        DCGA WTAB
        --LOCAL VARIABLES--
        DCLC ITNAM,TESTW
        --EXTENSION OF ITEM NAME TO 8 CHARACTERS--
$$ITNAM= SETC $ITNAME
$$ITNAM= SETC MOVE(00,1,2,7)
        --SUBROUTINE CALL FOR PROCESSING, AND SUPPLYING WORDS--
        SU WWWTAB. --SUBROUTINE CALL FOR PROCESSING--
        AA $TABNAM --SUPPLYING WORD (1ST GENE. DOPE VECT.)--
        '$$ITNAM' --SUPPLYING WORD (ITEM NAME 8 CHARACT.)--
        --''SECOND'' SUPPLYING WORD PART--
        --TEST IF ENTRY ORDER IS A DOPE VECTOR--
$$TESTW= SETC MOVE($IENTR,1,1,1)
        CAB ($$TESTW NE W) .OK1
        AA $IENTR --SUPPLYING WORD (ENTRY ORDER)--
        UAB .OK2
.OK1 = AA $IENTR --SUPPLYING WORD (ENTRY ORDER)--
.OK2 = AA $WDOPEV --SUPPLYING WORD (1ST TARGET DOPE VECT)--
        AA W&$WDOPEV --SUPPLYING WORD (2ND TARGET DOPE VECT)--
        --HERE IS THE SUBROUTINE RETURN--
        --TEST IF THE ITEM IS AN ARRAY--
        CAB ($IELEM EQ ' ') .OK3
        --IF YES, FILLING OF DOPE VECT. FOR THE ELEMENT--
        *ARRAY $WDOPEV,$IELEM,$WDOPEV
        --INITIALIZATION FOR SUBROUTINE GENERATION--
.OK3 = NOP
```

\$\$WTAB = SETA 1
MEND

```

MACRO
$LABEL = ASSIGN $RES,$V,$OP1,$V1,$OP2,$V2,$OP3,$V3, *
           $OP4,$V4,$OP5,$V5,$OP6,$V6
           --DECLARATION OF INTERNAL VARIABLES--
           --LOCAL VARIABLES--
           DCLA COUNT
           DCLC VAR,OP,TESTW
           --TEST IF 1ST VARIABLE IS A DOPE VECTOR--
$$TESTW= SETC MOVE($V,1,1,1)
           CAB ($$TESTW EQ W).OKO
           B $V --BRING 1ST VARIABLE--
           UAB .L1
           .OKO = BE $V --BRING 1ST VARIABLE--
           .L1 = NOP
           --INCREMENTATION OF THE COUNTER--
$$COUNT= SETA $$COUNT+1
           --TEST FOR NEXT VARIABLE PROCESSING--
           CAB ($$COUNT GT 1) .L2
           --TEST FOR END OF EXPRESSION--
           CAB ($OP1 EQ ' ') .STORE
           --INITIALIZATION OF INTERNAL VARIABLES--
$$OP = SETC $OP1
$$VAR = SETC $V1
           --JUMP TO PROCESSING PART--
           UAB .SEAROP
           --TEST FOR NEXT VARIABLE PROCESSING--
           .L2 = CAB ($$COUNT GT 2) .L3
           --TEST FOR END OF EXPRESSION--
           CAB ($OP2 EQ ' ') .STORE
           --INITIALIZATION OF INTERNAL VARIABLES--
$$OP = SETC $OP2
$$VAR = SETC $V2
           --JUMP TO PROCESSING PART--
           UAB .SEAROP
           --TEST FOR NEXT VARIABLE PROCESSING--
           .L3 = CAB ($$COUNT GT 3) .L4
           --TEST FOR END OF EXPRESSION--
           CAB ($OP3 EQ ' ') .STORE
           --INITIALIZATION OF INTERNAL VARIABLES--
$$OP = SETC $OP3
$$VAR = SETC $V3
           --JUMP TO PROCESSING PART--
           UAB .SEAROP
           --TEST FOR NEXT VARIABLE PROCESSING--
           .L4 = CAB ($$COUNT GT 4) .L5
           --TEST FOR END OF EXPRESSION--
           CAB ($OP4 EQ ' ') .STORE
           --INITIALIZATION OF INTERNAL VARIABLES--
$$OP = SETC $OP4
$$VAR = SETC $V4
           --JUMP TO PROCESSING PART--
           UAB .SEAROP
           --TEST FOR NEXT VARIABLE PROCESSING--
           .L5 = CAB ($$COUNT GT 5) .L6
           --TEST FOR END OF EXPRESSION--
           CAB ($OP5 EQ ' ') .STORE
           --INITIALIZATION OF INTERNAL VARIABLES--
$$OP = SETC $OP5
$$VAR = SETC $V5
           --JUMP TO PROCESSING PART--
           UAB .SEAROP
           --TEST FOR END OF EXPRESSION--
           .L6 = CAB ($$COUNT NE 6) .STORE
           --TEST FOR END OF EXPRESSION--
           CAB ($OP6 EQ ' ') .STORE
           --INITIALIZATION OF INTERNAL VARIABLES--
$$OP = SETC $OP6
$$VAR = SETC $V6
           --PROCESSING PART,RESEARCH OF RIGHT OPERATOR--
           --TEST FOR ADDITION--
           .SEAROP= CAB ($$OP NE '+').MINUS
           --TEST IF THE VARIABLE IS A DOPE VECTOR--
$$TESTW= SETC MOVE($$VAR,1,1,1)
           CAB ($$TESTW EQ W).OK1
           A $$VAR --ADDITION OF THE VARIABLE--

```

```

      UAB      .L1
.OK1  = AE      $$VAR      --ADDITION OF THE VARIABLE--
      UAB      .L1
      --TEST FOR SUBTRACTION--
.MINUS = CAB      ($$OP NE '-').MULT
      --TEST IF THE VARIABLE IS A DOPE VECTOR--
$$TESTW= SETC      MOVE($$VAR,1,1,1)
      CAB      ($$TESTW EQ W) .OK2
      SB      $$VAR      --SUBTRACTION OF THE VARIABLE--
      UAB      .L1
.OK2  = SBE      $$VAR      --SUBTRACTION OF THE VARIABLE--
      UAB      .L1
      --TEST FOR MULTIPLICATION--
.MULT  = CAB      ($$OP NE '*').DIV
      --TEST IF THE VARIABLE IS A DOPE VECTOR--
$$TESTW= SETC      MOVE($$VAR,1,1,1)
      CAB      ($$TESTW EQ W) .OK3
      ML      $$VAR      --MULTIPLICATION BY THE VARIABLE--
      SH      LZ      23  --(RESULT IN ACCUMULATOR)--
      UAB      .L1
.OK3  = MLE      $$VAR      --MULTIPLICATION BY THE VARIABLE--
      SH      LZ      23  --(RESULT IN ACCUMULATOR)--
      UAB      .L1
      --TEST FOR DIVISION--
.DIV   = CAB      ($$OP NE '/').ERROR
      --TEST IF THE VARIABLE IS A DOPE VECTOR--
$$TESTW= SETC      MOVE($$VAR,1,1,1)
      CAB      ($$TESTW EQ W) .OK4
      SU      DIVIDE.  --SUBROUTINE CALL FOR DIVISION PER--
      AA      $$VAR      --THE VARIABLE--
      UAB      .L1
.OK4  = SU      DIVIDE.  --SUBROUTINE CALL FOR DIVISION PER--
      AAE      $$VAR      --THE VARIABLE--
      UAB      .L1
.ERROR = NOTE      ILLEGAL OPERATOR IN THE PARAMETER LIST
      UAB      .L1
.STORE = NOP
      --TEST IF THE VARIABLE IS A DOPE VECTOR--
$$TESTW= SETC      MOVE($RES,1,1,1)
      CAB      ($$TESTW EQ W) .OK5
      C      $RES      --STORAGE OF THE RESULT--
      UAB      .END
.OK5  = CE      $RES      --STORAGE OF THE RESULT--
.END  = MEND

```

```

MACRO
BCHAR $NAME,$IELEM,$N
      --DECLARATION OF INTERNAL VARIABLES--
      --GLOBAL VARIABLES--
DCGA WBCHAR,WCHECK
      --LOCAL VARIABLE--
DCLC TESTW
      --FILLING OF SUPPLYING WORDS FOR THE SUBROUTINE--
      --FILLING OF 1ST SUPPLYING WORD--
B      $NAME      --FILLING OF 1ST SUPPLYING WORD AND--
C      WWWAO.      --STORAGE IN AN INTERNAL WORK VARIABLE--
      --FILLING OF 2ND SUPPLYING WORD--
      --TEST IF CHARACTER ORDER IS A DOPE VECTOR--
$$TESTW= SETC      MOVE($IELEM,1,1,1)
      CAB      ($$TESTW NE W) .OK1
$$TESTW= SETC      MOVE($IELEM,1,3,1)
      CAB      ($$TESTW EQ WWW) .OK1
      BE      $IELEM      --FILLING OF 2ND SUPPLYING WORD AND--
      UAB      .OK2
.OK1  = B      $IELEM      --FILLING OF 2ND SUPPLYING WORD AND--
.OK2  = C      WWWA1.      --STORAGE IN AN INTERNAL WORK VARIABLE--
      --FILLING OF 3RD SUPPLYING WORD--
      --TEST OF THE EXISTENCE OF CHARACTER NUMBER--
      CAB      ($N NE ' ') .OK3
      B      1      --FILLING OF 3RD SUPPLYING WORD AND--
      UAB      .OK5
      --TEST IF CHARACTER NUMBER IS A DOPE VECTOR--
.OK3  = NOP
$$TESTW= SETC      MOVE($N,1,1,1)
      CAB      ($$TESTW NE W) .OK4
$$TESTW= SETC      MOVE($N,1,3,1)
      CAB      ($$TESTW EQ WWW) .OK4
      BE      $N      --FILLING OF 3RD SUPPLYING WORD AND--

```

```

      UAB      .OK5
.OK4   = B     $N          --FILLING OF 3RD SUPPLYING WORD AND--
.OK5   = C     WWA2.       --STORAGE IN AN INTERNAL WORK VARIABLE--
      SU      WWWBCH.      --SUBROUTINE CALL FOR PROCESSING--
--INITIALIZATION FOR SUBROUTINE GENERATION--
$$WBCHAR=SETA 1
      MEND

```

```

MACRO
CCHAR $NAME,$IELEM,$N
--DECLARATION OF INTERNAL VARIABLES--
--GLOBAL VARIABLES
DCGA WCCHAR,WCHECK
--LOCAL VARIABLE--
DCLC TESTW
C      WWWBO. --STORAGE IN AN INTERNAL WORK VARIABLE--
--FILLING OF SUPPLYING WORDS FOR THE SUBROUTINE--
--FILLING OF 1ST SUPPLYING WORD--
B      $NAME   --FILLING OF 1ST SUPPLYING WORD AND--
C      WWA0.   --STORAGE IN AN INTERNAL WORK VARIABLE--
--FILLING OF 2ND SUPPLYING WORD--
--TEST IF CHARACTER ORDER IS A DOPE VECTOR--
$$TESTW=SETC MOVE($IELEM,1,1,1)
CAB ($$TESTW NE W) .OK1
$$TESTW= SETC MOVE($IELEM,1,3,1)
CAB ($$TESTW EQ WWW) .OK1
BE $IELEM --FILLING OF 2ND SUPPLYING WORD AND--
UAB .OK2
.OK1   = B     $IELEM --FILLING OF 2ND SUPPLYING WORD AND--
.OK2   = C     WWA1.  --STORAGE IN AN INTERNAL WORK VARIABLE--
--FILLING OF 3RD SUPPLYING WORD--
--TEST OF THE EXISTENCE OF CHARACTER NUMBER--
CAB ($N NE ' ') .OK3
B      1      --FILLING OF 3RD SUPPLYING WORD AND--
UAB .OK5
--TEST IF CHARACTER NUMBER IS A DOPE VECTOR--
.OK3   = NOP
$$TESTW= SETC MOVE($N,1,1,1)
CAB ($$TESTW NE W) .OK4
$$TESTW= SETC MOVE($N,1,3,1)
CAB ($$TESTW EQ WWW) .OK4
BE $N      --FILLING OF 3RD SUPPLYING WORD AND--
UAB .OK5
.OK4   = B     $N      --FILLING OF 3RD SUPPLYING WORD AND--
.OK5   = C     WWA2.   --STORAGE IN AN INTERNAL WORK VARIABLE--
      SU      WWWCCH.  --SUBROUTINE CALL FOR PROCESSING--
$$WCCHAR=SETA 1
      MEND

```

```

MACRO
$LAB= BBIT $NAME,$IELEM,$N~
--DECLARATION OF INTERNAL VARIABLES--
--GLOBAL VARIABLES--
DCGA WBBIT,WCHECK
--LOCAL VARIABLE--
DCLC TESTW
--FILLING OF SUPPLYING WORDS FOR THE SUBROUTINE--
--FILLING OF 1ST SUPPLYING WORD--
$LAB= B      $NAME   --FILLING OF 1ST SUPPLYING WORD AND--
C      WWA0.   --STORAGE IN AN INTERNAL WORK VARIABLE--
--FILLING OF 2ND SUPPLYING WORD--
--TEST IF BIT ORDER IS A DOPE VECTOR--
$$TESTW= SETC MOVE($IELEM,1,1,1)
CAB ($$TESTW NE W) .OK1
$$TESTW= SETC MOVE($IELEM,1,3,1)
CAB ($$TESTW EQ WWW) .OK1
BE $IELEM --FILLING OF 2ND SUPPLYING WORD AND--
UAB .OK2
.OK1   = B     $IELEM --FILLING OF 2ND SUPPLYING WORD AND--
.OK2   = C     WWA1.  --STORAGE IN AN INTERNAL WORK VARIABLE--
--FILLING OF 3RD SUPPLYING WORD--
--TEST OF THE EXISTENCE OF BIT NUMBER--
CAB ($N NE ' ') .OK3
B      1      --FILLING OF 3RD SUPPLYING WORD AND--
UAB .OK5
--TEST IF BIT NUMBER IS A DOPE VECTOR--

```

```

.OK3 = NOP
$$TESTW= SETC MOVE($N,1,1,1)
CAB ($$TESTW NE W) .OK4
$$TESTW= SETC MOVE($N,1,3,1)
CAB ($$TESTW EQ WWW) .OK4
BE $N --FILLING OF 3RD SUPPLYING WORD AND--
UAB .OK5
.OK4 = B $N --FILLING OF 3RD SUPPLYING WORD AND--
.OK5 = C WWWA2. --STORAGE IN AN INTERNAL WORK VARIABLE--
SU WWBB1. --SUBROUTINE CALL FOR PROCESSING--
--INITIALIZATION FOR SUBROUTINE GENERATION--
$$WBBIT= SETA 1
MEND

```



```

MACRO
$LABEL = MOVB $STR1,$I1,$STR2,$I2,$LENGTH
--DECLARATION OF INTERNAL VARIABLES--
--GLOBAL VARIABLE--
DCGA WMOVB
--LOCAL VARIABLE--
DCLC TESTW
--FILLING OF SUPPLYING WORDS FOR THE SUBROUTINE--
--FILLING OF 1ST SUPPLYING WORD--
$LABEL = B $STR1 --FILLING OF 1ST SUPPLYING WORD AND--
C WWWC1. --STORAGE IN AN INTERNAL WORK VARIABLE--
--FILLING OF 2ND SUPPLYING WORD--
--TEST IF 1ST BIT ORDER IS A DOPE VECTOR--
$$TESTW= SETC MOVE($I1,1,1,1)
CAB ($$TESTW EQ W) .OK1
B $I1 --FILLING OF 2ND SUPPLYING WORD AND--
UAB .OK2
.OK1 = BE $I1 --FILLING OF 2ND SUPPLYING WORD AND--
.OK2 = C WWWC2. --STORAGE IN AN INTERNAL WORK VARIABLE--
--FILLING OF 3RD SUPPLYING WORD--
B $STR2 --FILLING OF 3RD SUPPLYING WORD AND--
C WWWC3. --STORAGE IN AN INTERNAL WORK VARIABLE--
--FILLING OF 4TH SUPPLYING WORD--
--TEST IF 2ND BIT ORDER IS A DOPE VECTOR--
$$TESTW= SETC MOVE($I2,1,1,1)
CAB ($$TESTW EQ W) .OK3
B $I2 --FILLING OF 4TH SUPPLYING WORD AND--
UAB .OK4
.OK3 = BE $I2 --FILLING OF 4TH SUPPLYING WORD AND--
.OK4 = C WWWC4. --STORAGE IN AN INTERNAL WORK VARIABLE--
--FILLING OF 5TH SUPPLYING WORD--
--TEST IF THE LENGTH IS A DOPE VECTOR--
$$TESTW= SETC MOVE($LENGTH,1,1,1)
CAB ($$TESTW EQ W) .OK5
B $LENGTH --FILLING OF 5TH SUPPLYING WORD AND--
UAB .OK6
.OK5 = BE $LENGTH --FILLING OF 5TH SUPPLYING WORD AND--
.OK6 = C WWWC5. --STORAGE IN AN INTERNAL WORK VARIABLE--
SU WWMBV. --*SUBROUTINE CALL FOR PROCESSING*--
--INITIALIZATION FOR SUBROUTINE GENERATION--
$$WMOVB= SETA 1
MEND

```

```

MACRO
$LAB = CBIT $NAME,$IELEM,$N
--DECLARATION OF INTERNAL VARIABLES--
--GLOBAL VARIABLES--
DCGA WCBIT,WCHECK
--LOCAL VARIABLE--
DCLC TESTW
$LAB = C WWWB0. --STORAGE IN AN INTERNAL WORK VARIABLE--
--FILLING OF SUPPLYING WORDS FOR THE SUBROUTINE--
--FILLING OF 1ST SUPPLYING WORD--
B $NAME --FILLING OF 1ST SUPPLYING WORD AND--
C WWWA0. --STORAGE IN AN INTERNAL WORK VARIABLE--
--FILLING OF 2ND SUPPLYING WORD--
--TEST IF BIT ORDER IS A DOPE VECTOR--
$$TESTW=SETC MOVE($IELEM,1,1,1)
CAB ($$TESTW NE W) .OK1
$$TESTW= SETC MOVE($IELEM,1,3,1)
CAB ($$TESTW EQ WWW) .OK1
BE $IELEM --FILLING OF 2ND SUPPLYING WORD AND--

```

```

      UAB      .OK2
.OK1   = B      $IELEM      --FILLING OF 2ND SUPPLYING WORD AND--
.OK2   = C      WWWA1.      --STORAGE IN AN INTERNAL WORK VARIABLE--
                                --FILLING OF 3RD SUPPLYING WORD--
                                --TEST OF THE EXISTENCE OF BIT NUMBER--
      CAB      ($N NE ' ') .OK3
      B        1            --FILLING OF 3RD SUPPLYING WORD AND--
      UAB      .OK5
                                --TEST IF BIT NUMBER IS A DOPE VECTOR--
.OK3   = NOP
$$TESTW= SETC   MOVE($N,1,1,1)
      CAB      ($$TESTW NE W) .OK4
$$TESTW= SETC   MOVE($N,1,3,1)
      CAB      ($$TESTW EQ WWW) .OK4
      BE       $N            --FILLING OF 3RD SUPPLYING WORD AND--
      UAB      .OK5
.OK4   = B      $N            --FILLING OF 3RD SUPPLYING WORD AND--
.OK5   = C      WWWA2.      --STORAGE IN AN INTERNAL WORK VARIABLE--
      SU       WWWCB1.      --SUBROUTINE CALL FOR PROCESSING--
                                --INITIALIZATION FOR SUBROUTINE GENERATION--
$$WCBIT= SETA   1
      MEND

```

```

      MACRO
$LABEL = MOVC   $STR1,$I1,$STR2,$I2,$LENGTH
                                --DECLARATION OF INTERNAL VARIABLES--
                                --GLOBAL VARIABLE--
      DCGA      WMOVC
                                --LOCAL VARIABLE--
      DCLC      TESTW
                                --FILLING OF SUPPLYING WORDS FOR THE SUBROUTINE--
                                --FILLING OF 1ST SUPPLYING WORD--
$LABEL = B      $STR1      --FILLING OF 1ST SUPPLYING WORD AND--
      C        WWWC1.      --STORAGE IN AN INTERNAL WORK VARIABLE--
                                --FILLING OF 2ND SUPPLYING WORD--
                                --TEST IF 1ST CHAR. ORDER IS A DOPE VECT.--
$$TESTW= SETC   MOVE($I1,1,1,1)
      CAB      ($$TESTW EQ W) .OK1
      B        $I1        --FILLING OF 2ND SUPPLYING WORD AND--
      UAB      .OK2
.OK1   = BE     $I1        --FILLING OF 2ND SUPPLYING WORD AND--
.OK2   = C      WWWC2.      --STORAGE IN AN INTERNAL WORK VARIABLE--
                                --FILLING OF 3RD SUPPLYING WORD--
      B        $STR2      --FILLING OF 3RD SUPPLYING WORD AND--
      C        WWWC3.      --STORAGE IN AN INTERNAL WORK VARIABLE--
                                --FILLING OF 4TH SUPPLYING WORD--
                                --TEST IF 2ND CHAR. ORDER IS A DOPE VECT.--
$$TESTW= SETC   MOVE($I2,1,1,1)
      CAB      ($$TESTW EQ W) .OK3
      B        $I2        --FILLING OF 4TH SUPPLYING WORD AND--
      UAB      .OK4
.OK3   = BE     $I2        --FILLING OF 4TH SUPPLYING WORD AND--
.OK4   = C      WWWC4.      --STORAGE IN AN INTERNAL WORK VARIABLE--
                                --FILLING OF 5TH SUPPLYING WORD--
                                --TEST IF THE LENGTH IS A DOPE VECTOR--
$$TESTW= SETC   MOVE($LENGTH,1,1,1)
      CAB      ($$TESTW EQ W) .OK5
      B        $LENGTH    --FILLING OF 5TH SUPPLYING WORD AND--
      UAB      .OK6
.OK5   = BE     $LENGTH    --FILLING OF 5TH SUPPLYING WORD AND--
.OK6   = C      WWWC5.      --STORAGE IN AN INTERNAL WORK VARIABLE--
      SU       WWWMVC.      --*SUBROUTINE CALL FOR PROCESSING*--
                                --INITIALIZATION FOR SUBROUTINE GENERATION--
$$WMOVC= SETA   1
      MEND

```

```

      MACRO
      BOOL      $STR1,$I1,$OPER,$STR2,$I2,$LENGTH
                                --DECLARATION OF INTERNAL VARIABLES--
                                --GLOBAL VARIABLES--
      DCGA      WBOOL,WMB1,WMB3
                                --LOCAL VARIABLE--
      DCLC      TESTW
                                --FILLING OF SUPPLYING WORDS FOR THE SUBROUTINE--
                                --FILLING OF 1ST SUPPLYING WORD--
      B        $STR1      --FILLING OF 1ST SUPPLYING WORD AND--

```

```

C      WWWC1.      --STORAGE IN AN INTERNAL WORK VARIABLE--
                --FILLING OF 2ND SUPPLYING WORD--
                --TEST IF BIT ORDER IS A DOPE VECTOR--
$$TESTW= SETC    MOVE($I1,1,1,1)
CAB          ($$TESTW EQ W) .OK1
B            $I1      --FILLING OF 2ND SUPPLYING WORD AND--
UAB          .OK2
.OK1 = BE      $I1      --FILLING OF 2ND SUPPLYING WORD AND--
.OK2 = C      WWWC2.    --STORAGE IN AN INTERNAL WORK VARIABLE--
                --FILLING OF 3RD SUPPLYING WORD--
B            $STR2     --FILLING OF 3RD SUPPLYING WORD AND--
C            WWWC3.    --STORAGE IN AN INTERNAL WORK VARIABLE--
                --FILLING OF 4TH SUPPLYING WORD--
                --TEST IF BIT ORDER IS A DOPE VECTOR--
$$TESTW= SETC    MOVE($I2,1,1,1)
CAB          ($$TESTW EQ W) .OK3
B            $I2      --FILLING OF 4TH SUPPLYING WORD AND--
UAB          .OK4
.OK3 = BE      $I2      --FILLING OF 4TH SUPPLYING WORD AND--
.OK4 = C      WWWC4.    --STORAGE IN AN INTERNAL WORK VARIABLE--
                --FILLING OF 5TH SUPPLYING WORD--
                --TEST IF THE LENGTH IS A DOPE VECTOR--
$$TESTW= SETC    MOVE($LENGTH,1,1,1)
CAB          ($$TESTW EQ W) .OK5
B            $LENGTH  --FILLING OF 5TH SUPPLYING WORD AND--
UAB          .OK6
.OK5 = BE      $LENGTH  --FILLING OF 5TH SUPPLYING WORD AND--
.OK6 = C      WWWC5.    --STORAGE IN AN INTERNAL WORK VARIABLE--
                --FILLING OF 6TH SUPPLYING WORD--
                --TEST TO DIFFERENCIATE THE OPERATORS--
CAB          ($OPER NE AND) .OK8
.OK7 = B      -1      --FILLING 6TH SUPPLYING WORD (OPER. AND)--
UAB          .OK10
.OK8 = CAB     ($OPER NE OR) .OK9
SH          U      0    --FILLING 6TH SUPPLYING WORD (OPER. OR)--
UAB          .OK10
.OK9 = CAB     ($OPER NE XOR) .ERROR
B            1      --FILLING 6TH SUPPLYING WORD (OPER. XOR)--
UAB          .OK10
.ERROR = NOTE  INVALID OPERATOR IN MACRO BOOL,AND IS ASSUMED
UAB          .OK7
.OK10 = C      WWWC6.    --STORAGE IN AN INTERNAL WORK VARIABLE--
SU          WWWB00.     --SUBROUTINE CALL FOR PROCESSING--
                --INITIALIZATION FOR SUBROUTINE GENERATION--
$$WBOOL= SETA   1
$$WMB1 = SETA   1
$$WMB3 = SETA   1
MEND

```

```

MACRO
$LAB = FOR      $I,$INIT,$STEP,$LIMIT
                --DECLARATION OF INTERNAL VARIABLES--
                --GLOBAL VARIABLES--
DCGA WWWF,WWN
DCGA WWN1,WWN2,WWN3,WWN4,WWN5
DCGA WWN6,WWN7,WWN8,WWN9,WWN10
                --INITIALIZATION OR INCREMENTATION OF VARIABLES--
                --TO CALCULATE THE NESTED CALL NUMBER--
$$WWF = SETA    $$WWF+1
$$WWN = SETA    $$WWN+1
                --TEST FOR NESTING GREATER THAN 10--
                --1ST NESTED CALL --
CAB          ($$WWN NE 1) .OK2
                --INITIALIZATION OF A VARIABLE--
$$WWN1 = SETA   $$WWF
UAB          .OK1
                --2ND NESTED CALL --
.OK2 = CAB      ($$WWN NE 2) .OK3
                --INITIALIZATION OF A VARIABLE--
$$WWN2 = SETA   $$WWF
UAB          .OK1
                --3RD NESTED CALL --
.OK3 = CAB      ($$WWN NE 3) .OK4
                --INITIALIZATION OF A VARIABLE--
$$WWN3 = SETA   $$WWF
UAB          .OK1
                --4TH NESTED CALL --
.OK4 = CAB      ($$WWN NE 4) .OK5

```

```

--INITIALIZATION OF A VARIABLE--
$$WWN4 = SETA  $$WWF
        UAB     .OK1
--5TH NESTED CALL --
.OK5    = CAB  ($$WWN NE 5) .OK6
--INITIALIZATION OF A VARIABLE--
$$WWN5 = SETA  $$WWF
        UAB     .OK1
--6TH NESTED CALL --
.OK6    = CAB  ($$WWN NE 6) .OK7
--INITIALIZATION OF A VARIABLE--
$$WWN6 = SETA  $$WWF
        UAB     .OK1
--7TH NESTED CALL --
.OK7    = CAB  ($$WWN NE 7) .OK8
--INITIALIZATION OF A VARIABLE--
$$WWN7 = SETA  $$WWF
        UAB     .OK1
--8TH NESTED CALL --
.OK8    = CAB  ($$WWN NE 8) .OK9
--INITIALIZATION OF A VARIABLE--
$$WWN8 = SETA  $$WWF
        UAB     .OK1
--9TH NESTED CALL --
.OK9    = CAB  ($$WWN NE 9) .OK10
--INITIALIZATION OF A VARIABLE--
$$WWN9 = SETA  $$WWF
        UAB     .OK1
--10TH NESTED CALL --
.OK10   = CAB  ($$WWN NE 10) .ERROR
--IF YES,INITIALIZATION OF VARIABLE--
$$WWN10= SETA  $$WWF
        UAB     .OK1
--IF NO,ERROR MESSAGE--
.ERROR  = NOTE  MORE THAN 10 NESTED FOR LOOPS
        UAB     .END
--PROCESSING PART,TEST FOR THE END OF THE LOOP--
.OK1    = NOP
$LAB    = B     $INIT      --BRING INITIAL VALUE--
        S     3L
WWNB&$$WWF = B     $I      --BRING LOOP VARIABLE--
        A     $STEP      --ADD THE STEP--
        C     $I         --TO HAVE THE NEW VALUE OF THE LOOP VAR--
        SB    $LIMIT      --TEST FOR END OF THE LOOP--
        SK    3L         --IF NO,EXECUTE ONE MORE TIME THE--
        SI    2L         --INSTRUCTIONS IN THE LOOP--
        S     WWNB&$$WWF+1 --IF YES,RETURN JUMP OR END JUMP--
.END    = MEND

```

```

MACRO
$LAB    = END
--DECLARATION OF INTERNAL VARIABLES--
--GLOBAL VARIABLES--
DCGA    WVN,WENDO
DCGA    WVN1,WWN2,WWN3,WWN4,WWN5
DCGA    WVN6,WWN7,WWN8,WWN9,WWN10
DCLC    C,AUXL
--TEST FOR END OF NESTING OR END OF LOOP--
CAB     ($$WWN EQ 0) .ENDPRO
CAB     ($LAB EQ ' ') .ONEL
$$AUXL  = SETC  $LAB&'='
.ONEL   = NOP
$$C     = SETC  $$AUXL&S
        UAB     .NEST1
--IF END CASE,CALL OF END MACROS--
.ENDPRO= CAB    ($$WENDO EQ 0) .WENDCA
--IF NO,ERROR MESSAGE--
NOTE    INEFFECTIVE PROGRAM END.THE PROGRAM WAS CLOSED
NOTE    BY A PRECEDENT END MACRO
        UAB     .END
.WENDCA= NOP
$LAB    = SUEY START
$$WENDO= SETA  1
        *WEND1
        *WEND2
        *WEND3
        *WEND4
        *WEND5

```

```

*WEND6
*WEND7
UAB .END
--TEST FOR END OF LOOP (END 1ST OR BIGGER LOOP)--
.NEST1 = CAB ($$WWN NE 1) .NEST2
--IF NO,RETURN ON THE BEGINNING OF THE 1ST LOOP--
WWWB$$WWN1 = $$C WWWB$$WWN1 --RETURN JUMP--
UAB .NESTO
--TEST FOR END OF 2ND LOOP--
.NEST2 = CAB ($$WWN NE 2) .NEST3
--IF NO,RETURN ON THE BEGINNING OF THE 2ND LOOP--
WWWB$$WWN2 = $$C WWWB$$WWN2 --RETURN JUMP--
UAB .NESTO
--TEST FOR END OF 3RD LOOP--
.NEST3 = CAB ($$WWN NE 3) .NEST4
--IF NO,RETURN ON THE BEGINNING OF THE 3RD LOOP--
WWWB$$WWN3 = $$C WWWB$$WWN3 --RETURN JUMP--
UAB .NESTO
--TEST FOR END OF 4TH LOOP--
.NEST4 = CAB ($$WWN NE 4) .NEST5
--IF NO,RETURN ON THE BEGINNING OF THE 4TH LOOP--
WWWB$$WWN4 = $$C WWWB$$WWN4 --RETURN JUMP--
UAB .NESTO
--TEST FOR END OF 5TH LOOP--
.NEST5 = CAB ($$WWN NE 5) .NEST6
--IF NO,RETURN ON THE BEGINNING OF THE 5TH LOOP--
WWWB$$WWN5 = $$C WWWB$$WWN5 --RETURN JUMP--
UAB .NESTO
--TEST FOR END OF 6TH LOOP--
.NEST6 = CAB ($$WWN NE 6) .NEST7
--IF NO,RETURN ON THE BEGINNING OF THE 6TH LOOP--
WWWB$$WWN6 = $$C WWWB$$WWN6 --RETURN JUMP--
UAB .NESTO
--TEST FOR END OF 7TH LOOP--
.NEST7 = CAB ($$WWN NE 7) .NEST8
--IF NO,RETURN ON THE BEGINNING OF THE 7TH LOOP--
WWWB$$WWN7 = $$C WWWB$$WWN7 --RETURN JUMP--
UAB .NESTO
--TEST FOR END OF 8TH LOOP--
.NEST8 = CAB ($$WWN NE 8) .NEST9
--IF NO,RETURN ON THE BEGINNING OF THE 8TH LOOP--
WWWB$$WWN8 = $$C WWWB$$WWN8 --RETURN JUMP--
UAB .NESTO
--TEST FOR END OF 9TH LOOP--
.NEST9 = CAB ($$WWN NE 9) .NEST10
--IF NO,RETURN ON THE BEGINNING OF THE 9TH LOOP--
WWWB$$WWN9 = $$C WWWB$$WWN9 --RETURN JUMP--
UAB .NESTO
--TEST FOR END OF 10TH LOOP OR FOR AN ERROR--
.NEST10 = CAB ($$WWN NE 10) .ERROR
--IF YES,RETURN ON THE BEGINNING OF THE 10TH LOOP--
WWWB$$WWN10 = $$C WWWB$$WWN10 --RETURN JUMP--
UAB .NESTO
.ERROR = NOTE NESTING GREATER THAN 10,NO GENERATION CODE
UAB .NESTO
--DECREMENTATION OF THE LOOP NUMBER--
.NESTO = NOP
$$WWN = SETA $$WWN-1
.END = MEND

```

```

$$LABEL = MACRO
    IFA $VAR1, $COND, $VAR2, $LABO, $LABI
    MACRO
        --DECLARATION OF INTERNAL VARIABLES--
        --LOCAL VARIABLES--
        DCLC WIND1,WIND2,WS,TESTW
        UCLA W21,WNOT
        --TEST IF THE 1ST VARIABLE IS A DOPE VECTOR--
        $$TESTW = SETC MOVE($VAR1,1,1,1)
        CAB ($$TESTW NE W) .OK1
        --IF YES,SETTING OF THE INDIRECT SPECIFICATION--
        $$WIND1 = SETC E
        --TEST IF THE 2ND VARIABLE IS A DOPE VECTOR--
        .OK1 = NOP
        $$TESTW = SETC MOVE($VAR2,1,1,1)
        CAB ($$TESTW NE W) .EQ
        --IF YES,SETTING OF THE INDIRECT SPECIFICATION--
        $$WIND2 = SETC E
        --TEST OF THE CONDITIONS--
        --TEST FOR AN EQUAL CONDITION--

```

```

.EQ      = CAB      ($COND NE EQ) .GT
.EQUAL   = NOP
--IF YES,SET A JUMP--
$$WS     = SETC     SI
          UAB       .IFYES
--TEST FOR A GREATER THAN CONDITION--
.GT      = CAB      ($COND NE GT) .LT
--IF YES,SET A JUMP AND A SWITCH--
$$WS     = SETC     SK
$$W21    = SETA     1
          UAB       .IFYES
--TEST FOR A LESS THAN CONDITION--
.LT      = CAB      ($COND NE LT) .GE
--IF YES,SET A JUMP--
$$WS     = SETC     SK
          UAB       .IFYES
--TEST FOR A GREATER EQUAL CONDITION--
.GE      = CAB      ($COND NE GE) .LE
--IF YES,SET A JUMP--
$$WS     = SETC     SK
          UAB       .IFNOT
--TEST FOR A LESS EQUAL CONDITION--
.LE      = CAB      ($COND NE LE) .NE
--IF YES,SET A JUMP AND A SWITCH--
$$WS     = SETC     SK
$$W21    = SETA     1
          UAB       .IFNOT
--TEST FOR A NOT EQUAL CONDITION--
.NE      = CAB      ($COND NE NE) .ERROR
--IF YES,SET A JUMP--
$$WS     = SETC     SI
          UAB       .IFNOT
--IF NO,ERROR MESSAGE--
.ERROR   = NOTE     ILLEGAL CONDITION IN AN IFA MACRO.EQ IS ASSUMED
          UAB       .EQUAL
--PROCESSING AND GENERATION PART--
--SETTING OF A SWITCH FOR INVERS JUMP--
.IFNOT   = NOP
$$WNOT   = SETA     1
--TEST FOR THE SUBTRACTION OPERATION--
--FOR THE COMPARISON--
.IFYES   = NOP
          CAB      ($$W21 EQ 1) .OK2
          B&$$WIND1 $VAR1 --BRING 1ST VARIABLE AND SUBTRACT--
          SB&$$WIND2 $VAR2 --THE 2ND FOR THE COMPARISON--
          UAB      .OK3
.OK2     = B&$$WIND2 $VAR2 --BRING 2ND VARIABLE AND SUBTRACT--
          SB&$$WIND1 $VAR1 --THE 1ST FOR THE COMPARISON--
          --TEST OF A SWITCH FOR INVERS JUMP--
.OK3     = CAB      ($$WNOT NE 1) .OK6
          --IF YES,TEST IF A 2ND LABEL EXIST--
          CAB      ($LAB1 NE ' ') .OK4
          --IF NO,JUMP TO NEXT INSTRUCTION TO EXECUTE--
          $$WS     2L
          UAB      .OK5
          --IF YES,JUMP TO THIS LABEL--
.OK4     = $$WS     $LAB1 --JUMP TO 2ND LABEL,CONDITION UNFULFILLED--
.OK5     = S        $LAB0 --JUMP TO 1ST LABEL,CONDITION FULFILLED--
          UAB      .END
          --IF NO INVERS JUMP--
.OK6     = $$WS     $LAB0 --JUMP TO 1ST LABEL,CONDITION FULFILLED--
          --TEST IF A 2ND LABEL EXIST--
          CAB      ($LAB1 EQ ' ') .END
          --IF YES,JUMP TO 2ND LABEL--
          S        $LAB1 --JUMP TO 2ND LABEL,CONDITION UNFULFILLED--
.END     = MEND

MACRO
$LABEL = IFC  $STR1,$I1,$OPER,$STR2,$I2,$LENGTH,$LAB0,$LAB1
--DECLARATION OF INTERNAL VARIABLES--
--GLOBAL VARIABLES--
DCGA  WIFB,WBBI,WIFC
--LOCAL VARIABLE--
DCLC  TESTW
--FILLING OF SUPPLYING WORDS FOR THE SUBROUTINE--
--FILLING OF 1ST SUPPLYING WORD--
$LABEL = B    $STR1 --FILLING OF 1ST SUPPLYING WORD AND--
          C      WWWC1. --STORAGE IN AN INTERNAL WORK VARIABLE--

```

```

--FILLING OF 2ND SUPPLYING WORD--
--TEST IF CHAR ORDER IS A DOPE VECTOR--
$$TESTW= SETC MOVE($I1,1,1,1)
CAB ($$TESTW EQ W) .OK1
B $I1 --FILLING OF 2ND SUPPLYING WORD AND--
UAB .OK2
.OK1 = BE $I1 --FILLING OF 2ND SUPPLYING WORD AND--
.OK2 = C WWC2. --STORAGE IN AN INTERNAL WORK VARIABLE--
--FILLING OF 3RD SUPPLYING WORD--
B $STR2 --FILLING OF 3RD SUPPLYING WORD AND--
C WWC3. --STORAGE IN AN INTERNAL WORK VARIABLE--
--FILLING OF 4TH SUPPLYING WORD--
--TEST IF CHAR ORDER IS A DOPE VECTOR--
$$TESTW= SETC MOVE($I2,1,1,1)
CAB ($$TESTW EQ W) .OK3
B $I2 --FILLING OF 4TH SUPPLYING WORD AND--
UAB .OK4
.OK3 = BE $I2 --FILLING OF 4TH SUPPLYING WORD AND--
.OK4 = C WWC4. --STORAGE IN AN INTERNAL WORK VARIABLE--
--FILLING OF 5TH SUPPLYING WORD--
--TEST IF THE LENGTH IS A DOPE VECTOR--
$$TESTW= SETC MOVE($LENGTH,1,1,1)
CAB ($$TESTW EQ W) .OK5
B $LENGTH --FILLING OF 5TH SUPPLYING WORD AND--
UAB .OK6
.OK5 = BE $LENGTH --FILLING OF 5TH SUPPLYING WORD AND--
.OK6 = C WWC5. --STORAGE IN AN INTERNAL WORK VARIABLE--
--FILLING OF 6TH SUPPLYING WORD--
--TEST TO DIFFERENTIATE THE OPERATORS--
CAB ($OPER NE NE) .OK8
B -1 --FILLING 6TH SUPPLYING WORD (OPER. NE)--
UAB .OK10
.OK8 = CAB ($OPER NE EQ) .ERROR
.OK9 = SH V 0 --FILLING 6TH SUPPLYING WORD (OPER. EQ)--
UAB .OK10
.ERROR = NOTE INVALID OPERATOR IN MACRO IFC,EQ IS ASSUMED
UAB .OK9
.OK10 = C WWC0. --STORAGE IN AN INTERNAL WORK VARIABLE--
SU WWC1. --SUBROUTINE CALL FOR PROCESSING--
--FILLING OF 7TH SUPPLYING WORD WITH ADD OF 1ST LAB.--
S $LAB0 --CONDITION SATISFIED,JUMP TO 1ST LABEL--
--TEST IF 2ND LABEL EXIST--
CAB ($LAB1 EQ ' ') .OK11
--IF YES,FILLING OF 8TH SUPPLYING WORD--
--WITH THE ADDRESS OF THE 2ND LABEL--
S $LAB1 --COND. NOT SATISFIED,JUMP TO 2ND LABEL--
--INITIALIZATION FOR SUBROUTINE GENERATION--
.OK11 = NOP
$$WIFB = SETA 1
$$WBB1 = SETA 1
$$WIFC = SETA 1
MEND

```

```

MACRO
$LABEL = IFB $STR1,$I1,$OPER,$STR2,$I2,$LENGTH,$LAB0,$LAB1
--DECLARATION OF INTERNAL VARIABLES--
--GLOBAL VARIABLES--
DCGA WIFB,WBB1
--LOCAL VARIABLE--
DCLC TESTW
--TEST IF WE WANT JUST TO TEST ONE STRING--
CAB ($LENGTH NE ' ') .OK0
--IF YES,CALL MACRO TESTB,AND JUMP TO END--
$LABEL = *TESTB $STR1,$I1,$OPER,$STR2,$I2
UAB .END
--FILLING OF SUPPLYING WORDS FOR THE SUBROUTINE--
--FILLING OF 1ST SUPPLYING WORD--
.OK0 = NOP
$LABEL = B $STR1 --FILLING OF 1ST SUPPLYING WORD AND--
C WWC1. --STORAGE IN AN INTERNAL WORK VARIABLE--
--FILLING OF 2ND SUPPLYING WORD--
--TEST IF BIT ORDER IS A DOPE VECTOR--
$$TESTW= SETC MOVE($I1,1,1,1)
CAB ($$TESTW EQ W) .OK1
B $I1 --FILLING OF 2ND SUPPLYING WORD AND--
UAB .OK2
.OK1 = BE $I1 --FILLING OF 2ND SUPPLYING WORD AND--
.OK2 = C WWC2. --STORAGE IN AN INTERNAL WORK VARIABLE--

```

```

--FILLING OF 3RD SUPPLYING WORD--
B   $STR2   --FILLING OF 3RD SUPPLYING WORD--
C   WWC3.   --STORAGE IN AN INTERNAL WORK VARIABLE--
--FILLING OF 4TH SUPPLYING WORD--
--TEST IF BIT ORDER IS A DOPE VECTOR--
$$TESTW= SETC MOVE($I2,1,1,1)
CAB ($$TESTW EQ W) .OK3
B   $I2     --FILLING OF 4TH SUPPLYING WORD AND--
UAB .OK4
.OK3 = BE $I2     --FILLING OF 4TH SUPPLYING WORD AND--
.OK4 = C   WWC4.   --STORAGE IN AN INTERNAL WORK VARIABLE--
--FILLING OF 5TH SUPPLYING WORD--
--TEST IF THE LENGTH IS A DOPE VECTOR--
$$TESTW= SETC MOVE($LENGTH,1,1,1)
CAB ($$TESTW EQ W) .OK5
B   $LENGTH --FILLING OF 5TH SUPPLYING WORD AND--
UAB .OK6
.OK5 = BE $LENGTH --FILLING OF 5TH SUPPLYING WORD AND--
.OK6 = C   WWC5.   --STORAGE IN AN INTERNAL WORK VARIABLE--
--FILLING OF 6TH SUPPLYING WORD--
--TEST TO DIFFERENCIATE THE OPERATORS--
CAB ($OPER NE NE) .OK8
B   -1       --FILLING 6TH SUPPLYING WORD (OPER. NE)--
UAB .OK10
.OK8 = CAB ($OPER NE EQ) .ERROR
.OK9 = SH V 0   --FILLING 6TH SUPPLYING WORD (OPER. EQ)--
UAB .OK10
.ERROR = NOTE INVALID OPERATOR IN MACRO IFB,EQ IS ASSUMED
UAB .OK9
.OK10 = C   WWC0.   --STORAGE IN AN INTERNAL WORK VARIABLE--
SU   WWCIFB.   --SUBROUTINE CALL FOR PROCESSING--
--FILLING OF 7TH SUPPLYING WORD WITH ADD OF 1ST LAB.--
S   $LAB0     --CONDITION SATISFIED,JUMP TO 1ST LABEL--
--TEST IF 2ND LABEL EXIST--
CAB ($LAB1 EQ ' ') .OK11
--IF YES,FILLING OF 8TH SUPPLYING WORD--
--WITH THE ADDRESS OF THE 2ND LABEL--
S   $LAB1     --COND. NOT SATISFIED,JUMP TO 2ND LABEL--
.OK11 = NOP
--INITIALIZATION FOR SUBROUTINE GENERATION--
$$WIFB = SETA 1
$$WBB1 = SETA 1
.END = MEND

```

```

MACRO
$LABEL = TESTB $NAME,$I,$LENGTH,$LAB0,$LAB1
--DECLARATION OF INTERNAL VARIABLES--
--GLOBAL VARIABLES--
DCGA WTESTB,WBB1T
--LOCAL VARIABLE--
DCLC TESTW
--FILLING OF SUPPLYING WORDS FOR THE SUBROUTINE--
--FILLING OF 1ST SUPPLYING WORD--
$LABEL = B   $NAME     --FILLING OF 1ST SUPPLYING WORD--
C   WWA0.   --STORAGE IN AN INTERNAL WORK VARIABLE--
--FILLING OF 2ND SUPPLYING WORD--
--TEST IF THE BIT ORDER IS A DOPE VECTOR--
$$TESTW= SETC MOVE($I,1,1,1)
CAB ($$TESTW EQ W) .OK1
B   $I       --FILLING OF 2ND SUPPLYING WORD--
UAB .OK2
.OK1 = BE $I       --FILLING OF 2ND SUPPLYING WORD--
.OK2 = C   WWA1.   --STORAGE IN AN INTERNAL WORK VARIABLE--
--FILLING OF 3RD SUPPLYING WORD--
--TEST IF THE LENGTH IS A DOPE VECTOR--
$$TESTW= SETC MOVE($LENGTH,1,1,1)
CAB ($$TESTW EQ W) .OK3
B   $LENGTH --FILLING OF 3RD SUPPLYING WORD--
UAB .OK4
.OK3 = BE $LENGTH --FILLING OF 3RD SUPPLYING WORD--
.OK4 = C   WWA2.   --STORAGE IN AN INTERNAL WORK VARIABLE--
SU   WWTES.   --SUBROUTINE CALL FOR PROCESSING--
--FILLING OF 4TH SUPPLYING WORD WITH ADD OF 1ST LAB.--
S   $LAB0     --CONDITION SATISFIED,JUMP TO 1ST LABEL--
--TEST IF 2ND LABEL EXIST--
CAB ($LAB1 EQ ' ') .OK11
--IF YES,FILLING OF 5TH SUPPLYING WORD--
--WITH THE ADDRESS OF THE 2ND LABEL--

```

```

      S      $LAB1      --COND. NOT SATISFIED, JUMP TO 2ND LABEL--
.OK11 = NOP
      --INITIALIZATION FOR SUBROUTINE GENERATION--
$$WTESTB=SETA 1
$$WBBIT =SETA 1
      MEND

      MACRO
$LABEL = READ  $K1,$N1,$A1,$K2,$N2,$A2,$K3,$N3,$A3,$K4,$N4,$A4,
               $K5,$N5,$A5
      --DECLARATION OF INTERNAL VARIABLES--
      --GLOBAL VARIABLES--
      DCGA WIN1,WIN2,WOUT2,WHEXIN,WBININ
      --LOCAL VARIABLES--
      DCLC LABEL,K,N,A
      DCLA COUNT
      --INITIALIZATION FOR LABEL GENERATION--
$$LABEL= SETC $LABEL
      --INITIALIZATION FOR STANDARDIZING GENERATION--
$$WOUT2= SETA 0
      --TEST TO STANDARDIZE THE CARD READER--
      CAB ($$WIN1 EQ 1) .READ
$$WIN1 = SETA 1
$$LABEL= SUEY GENORM
               '1105'H
      AA ERROR1.
$$LABEL= SETC ' '
      --TEST FOR A NEW CHOICE OF IN/OUTPUT DEVICE--
      .READ = CAB ($$WIN2 EQ 1) .READ0
$$WIN2 = SETA 1
$$LABEL= SUEY EINVOR
               '5'H
      S -2L
$$LABEL= SETC ' '
      --INITIALIZATION FOR THE GENERAL PROCESSING--
      .READ0 = NOP
$$K = SETC $K1
$$N = SETC $N1
$$A = SETC $A1
      .READ00= CAB ($$N NE ' ') .READ1
$$N = SETC 1
      --TEST FOR END OF INPUT--
      .READ1 = CAB ($$K EQ ' ') .END
$$COUNT= SETA $$COUNT+1
      --TEST FOR CHARACTER INPUT--
      CAB ($$K NE C) .READ2
      .IFERR = NOP
$$LABEL= B $$A
      ET 'FFFF'H
      C WWWA1.
      B $$N
      SH L 16
      A WWWA1.
      C 2L
      SUEY EINSTR
      O
$$LABEL= SETC ' '
      --TEST FOR A RIGHT UPDATING OF THE VARIABLES--
      .INIT0 = CAB ($$COUNT EQ 1) .INIT1
      CAB ($$COUNT EQ 2) .INIT2
      CAB ($$COUNT EQ 3) .INIT3
      CAB ($$COUNT EQ 4) .INIT4
      CAB ($$COUNT EQ 5) .END
      --TEST FOR DECIMAL NUMBER INPUT--
      .READ2 = CAB ($$K NE D) .READ3
$$LABEL= B $$A
      ET 'FFFF'H
      C WWWA1.
      B $$N
      SH L 15
      AT '2805C0'H
      C 2L
      SUEY EINKVH
      O
      S ERROR3.
      CE WWWA1.
$$LABEL= SETC ' '
      UAB .INIT0

```

```

--TEST FOR HEXADECIMAL NUMBER INPUT--
.READ3 = CAB ($$K NE H) .READ4
$$LABEL= B $$A
          ET 'FFFF'H
          C WWA1.
          B $$N
          C WWA4.
$$WHEXIN=SETA 1
          SU WHEXIN.
$$LABEL= SETC ' '
          UAB .INITO

--TEST FOR BINARY NUMBER INPUT--
.READ4 = CAB ($$K NE B) .READ5
$$LABEL= B $$A
          ET 'FFFF'H
          C WWA1.
          B $$N
          C WWA4.
$$WBININ=SETA 1
          SU WBININ.
$$LABEL= SETC ' '
          UAB .INITO
.READ5 = CAB ($$K NE X) .READ6
$$LABEL= *FOR WWA1,1,1,$$N
          SUEY EINGAB
          *END
$$LABEL= SETC ' '
          UAB .INITO
.READ6 = CAB ($$K NE S) .ERROR
$$LABEL= SUEY BRIZEI
          '227'H
          S -2L
          ' '
$$LABEL= SETC ' '
          UAB .INITO
.INIT1 = NOP
$$K = SETC $K2
$$N = SETC $N2
$$A = SETC $A2
          UAB .READOO
.INIT2 = NOP
$$K = SETC $K3
$$N = SETC $N3
$$A = SETC $A3
          UAB .READOO
.INIT3 = NOP
$$K = SETC $K4
$$N = SETC $N4
$$A = SETC $A4
          UAB .READOO
.INIT4 = NOP
$$K = SETC $K5
$$N = SETC $N5
$$A = SETC $A5
          UAB .READOO
.ERROR = NOTE ILLEGAL KEY IN MACRO READ,C IS ASSUMED
          UAB .IFERR
.END = MEND

```

```

MACRO
$LABEL = PRINT $K1,$N1,$A1,$K2,$N2,$A2,$K3,$N3,$A3,
               $K4,$N4,$A4,$K5,$N5,$A5
--DECLARATION OF INTERNAL VARIABLES--
--GLOBAL VARIABLES--
DCGA WOUT1,WOUT2,WIN2,WHEXOU,WBINOU
--LOCAL VARIABLES--
DCLC LABEL,K,N,A
DCLA COUNT
--INITIALIZATION FOR LABEL GENERATION--
$$LABEL= SETC $LABEL
--INITIALIZATION FOR STANDARDIZING GENERATION--
$$WIN2 = SETA 0
--TEST TO STANDARDIZE THE PRINTER--
CAB ($$WOUT1 EQ 1) .PRIN
$$WOUT1= SETA 1
$$LABEL= SUEY GENORM
          '4'H
          AA ERROR2.
          ' '
$$LABEL= SETC ' '

```

*

```

--TEST FOR A NEW CHOICE OF IN/OUTPUT DEVICE--
.PRIN = CAB ($$WOUT2 EQ 1) .PRINO
$$WOUT2= SETA 1
$$LABEL= SUEY AUSVOR
          S -2L
          S -3L
          ' '
$$LABEL= SETC ' '
--INITIALIZATION FOR THE GENERAL PROCESSING--
.PRINO = NOP
$$K = SETC $K1
$$N = SETC $N1
$$A = SETC $A1
.PRIN00= CAB ($$N NE ' ') .PRIN1
$$N = SETC 1
--TEST FOR END OF OUTPUT
.PRIN1 = CAB ($$K EQ ' ') .END
$$COUNT= SETA $$COUNT+1
--TEST FOR CHARACTER OUTPUT--
CAB ($$K NE C) .PRIN2
$$LABEL= B $$A
          ET 'FFFF'H
          C WWA1.
          B $$N
          SH L 16
          A WWA1.
          C 2L
          SUEY AUSSTR
          0
$$LABEL= SETC ' '
--TEST FOR A RIGHT UPDATING OF THE VARIABLES--
.INITO = CAB ($$COUNT EQ 1) .INIT1
          CAB ($$COUNT EQ 2) .INIT2
          CAB ($$COUNT EQ 3) .INIT3
          CAB ($$COUNT EQ 4) .INIT4
          CAB ($$COUNT EQ 5) .END
--TEST FOR DECIMAL NUMBER OUTPUT--
.PRIN2 = CAB ($$K NE D) .PRIN3
$$LABEL= B $$A
          ET 'FFFF'H
          C WWA1.
          B $$N
          SH L 3
          AT '3270000'D
          C 3L
          BE WWA1.
          SUEY AUSKVZ
          0
$$LABEL= SETC ' '
          UAB .INITO
--TEST FOR HEXADECIMAL NUMBER OUTPUT--
.PRIN3 = CAB ($$K NE H) .PRIN4
$$LABEL= B $$A
          ET 'FFFF'H
          C WWA1.
          B $$N
          C WWA4.
$$WHEXOU=SETA 1
          SU WHEXOU.
          ' '
$$LABEL= SETC ' '
          UAB .INITO
--TEST FOR BINARY NUMBER OUTPUT--
.PRIN4 = CAB ($$K NE B) .PRIN5
$$LABEL= B $$A
          ET 'FFFF'H
          C WWA1.
          B $$N
          C WWA4.
$$WBINOU=SETA 1
          SU WBINOU.
          ' '
$$LABEL= SETC ' '
          UAB .INITO
.PRIN5 = CAB ($$K NE X) .PRIN6
$$LABEL= *FOR WWI,1,1,$$N
          B 42
          SUEY AUSGAB
          *END
          ' '
$$LABEL= SETC ' '
          UAB .INITO
.PRIN6 = CAB ($$K NE S) .ERROR

```

```

$$LABEL= B      39
          SUEY  AUSGAB
$$LABEL= SETC    ' '
          UAB    .INITO
.INIT1 = NOP
$$K     = SETC   $K2
$$N     = SETC   $N2
$$A     = SETC   $A2
          UAB    .PRIN00
.INIT2 = NOP
$$K     = SETC   $K3
$$N     = SETC   $N3
$$A     = SETC   $A3
          UAB    .PRIN00
.INIT3 = NOP
$$K     = SETC   $K4
$$N     = SETC   $N4
$$A     = SETC   $A4
          UAB    .PRIN00
.INIT4 = NOP
$$K     = SETC   $K5
$$N     = SETC   $N5
$$A     = SETC   $A5
          UAB    .PRIN00
.ERROR = NOTE   ILLEGAL KEY IN MACRO PRINT,NO OUTPUT
.END     = MEND

```

```

MACRO
$LABEL = READA $K,$N,$Z,$NX
          --DECLARATION OF INTERNAL VARIABLES--
          --GLOBAL VARIABLES--
          DCGA WIN1,WIN2,WOUT2
          --LOCAL VARIABLE--
          DCLC LABEL,N,NX
          --INITIALIZATION IF N EQUAL 0--
$$N      = SETC   $N
          CAB     ($$N NE ' ') .READA
$$N      = SETC   1
.READA = NOP
          --INITIALIZATION FOR THE LABEL GENERATION--
$$LABEL= SETC    $LABEL
          --INITIALIZATION FOR STANDARDIZING GENERATION--
$$WOUT2= SETA    0
          --TEST TO STANDARDIZE THE CARD READER--
          CAB     ($$WIN1 EQ 1) .READA1
$$WIN1 = SETA    1
$$LABEL= SUEY    GENORM
          '1105'H
          AA      ERROR1.
          ' '
$$LABEL= SETC    ' '
          --TEST FOR A NEW CHOICE OF IN/OUTPUT DEVICE--
.READA1= CAB     ($$WIN2 EQ 1) .READA2
$$WIN2 = SETA    1
$$LABEL= SUEY    EINVOR
          '5'H
          S       -2L
          ' '
$$LABEL= SETC    ' '
          --TEST FOR CHARACTER INPUT--
.READA2= CAB     ($K NE C) .READA3
.IFERR = NOP
$$LABEL= B       $$N
          SH      L      16
          AA      WWWI1.
          C       2L
          SUEY    EINSTR
          0
$$LABEL= SETC    ' '
          CAB     ($Z EQ ' ') .END
          UAB     .FORMAT
          --TEST FOR DECIMAL NUMBER INPUT--
.READA3= CAB     ($K NE D) .READA4
$$LABEL= B       $$N
          SH      L      15
          AT      '2805C0'H
          C       2L
          SUEY    EINKVH
          0
$$LABEL= SETC    ' '

```

```

S      ERROR3.
CAB    ($Z EQ ' ') .END
C      WWWI1.
UAB    .FORMAT
--TEST FOR HEXA-DECIMAL NUMBER INPUT--
. READA4= CAB    ($K NE H) .READA5
$$LABEL= *FOR    WWWI,1,1,$$N
SUEY    EINGAB
$$LABEL= SETC    ' '
SH      L      20
SH      ZLK    4
*END
SH      ZL      24
CAB    ($Z EQ ' ') .END
C      WWWI1.
UAB    .FORMAT
--TEST FOR BINARY NUMBER INPUT--
. READA5= CAB    ($K NE B) .ERROR
$$LABEL= *FOR    WWWI,1,1,$$N
SUEY    EINGAB
$$LABEL= SETC    ' '
SH      Z      1
*END
SH      ZL      24
SH      V      24
CAB    ($Z EQ ' ') .END
C      WWWI1.
. FORMAT= CAB    ($Z NE X) .READA7
$$NX    = SETC    $NX
CAB    ($$NX NE ' ') .READA6
$$NX    = SETC    1
. READA6= *FOR    WWWI,1,1,$$N
SUEY    EINGAB
*END
UAB    .END
. READA7= CAB    ($Z NE S) .END
SUEY    BRIZEI
        '227'H
S      -2L
UAB    .END
. ERROR = NOP
NOTE    ILLEGAL KEY IN MACRO READA,C IS ASSUMED
UAB    .IFERR
. END    = CAB    ($Z NE ' ') .BRING
CAB    ($K NE C) .ENDE
. BRING = 6      WWWI1.
. ENDE = MEND

MACRO
$$LABEL = PRINTA    $K,$N,$Z,$NX
--DECLARATION OF INTERNAL VARIABLES--
--GLOBAL VARIABLES--
DCGA    WOUT1,WOUT2,WIN2
DCLC    N,NX
--INITIALIZATION IF N EQUAL 0--
$$N      = SETC    $N
CAB    ($$N NE ' ') .PRINA
$$N      = SETC    1
. PRINA = NOP
--STORAGE OF THE OUTPUT--
$$LABEL = C      WWWOUT.
--INITIALIZATION FOR STANDARDIZING GENERATION--
$$WIN2 = SETA    0
--TEST TO STANDARDIZE THE PRINTER--
CAB    ($$WOUT1 EQ 1) .PRINA1
$$WOUT1= SETA    1
SUEY    GENORM
        '4'H
AA      ERROR2.
--TEST FOR A NEW CHOICE OF IN/OUTPUT DEVICE--
. PRINA1= CAB    ($$WOUT2 EQ 1) .PRINA2
$$WOUT2= SETA    1
SUEY    AUSVOR
        '000004'H
S      -2L
S      -3L
--TEST FOR CHARACTER OUTPUT--
. PRINA2= CAB    ($K NE C) .PRINA3

```

```

B      $$N
SH     L      16
AA     WWWOUT.
C      2L
SUEY   AUSSTR
O
UAB     .END
--TEST FOR DECIMAL NUMBER OUTPUT--
.PRINA3= CAB ($K NE D) .PRINA4
B      $$N
SH     L      3
AT     '03270000'D
C      3L
B      WWWOUT.
SUEY   AUSKVZ
O
UAB     .END
--TEST FOR HEXADECIMAL NUMBER OUTPUT--
.PRINA4= CAB ($K NE H) .PRINA5
B      $$N
SH     L      2
AT     '8000'H
C      WWWA1.
B      WWWOUT.
SHE    WWWA1.
SH     ZL     24
C      WWWOUT.
*FOR   WWWI,1,1,$$N
B      WWWOUT.
SH     V      0
SH     ZL     4
C      WWWA1.
SH     ZL     24
C      WWWOUT.
B      WWWA1.
SUEY   AUSGAB
*END
UAB     .END
--TEST FOR BINARY NUMBER OUTPUT--
.PRINA5= CAB ($K NE B) .ERROR
.IFERR = NOP
O      $$N
AT     '8000'H
C      WWWA1.
B      WWWOUT.
SHE    WWWA1.
SH     ZL     24
C      WWWOUT.
*FOR   WWWI,1,1,$$N
B      WWWOUT.
SH     V      0
SH     ZL     1
C      WWWA1.
SH     ZL     24
C      WWWOUT.
B      WWWA1.
SUEY   AUSGAB
*END
UAB     .END
.END    = CAB ($Z NE X) .PRINA7
$$NX    = SETC $NX
CAB     ($$NX NE ' ') .PRINA6
$$NX    = SETC 1
.PRINA6= *FOR WWWI,1,1,$$NX
B      42
SUEY   AUSGAB
*END
UAB     .ENDE
.PRINA7= CAB ($Z NE S) .ENDE
B      39
SUEY   AUSGAB
UAB     .ENDE
.ERROR = NOTE ILLEGAL KEY IN MACRO PRINTA,B IS ASSUMED
$$N    = SETC 24
UAB     .IFERR
.ENDE  = MEND

```

```

WEND1
--DECLARATION OF INTERNAL VARIABLES--
--GLOBAL VARIABLES--
DCGA  WTAB,WBCHAR
--TEST FOR SUBROUTINE GENERATION--
--FOR MACRO TABLE--
CAB    ($$WTAB NE 1) .OK1
WWWTAB.= 0      --SUBROUTINE FOR PROCESSING,MACRO TABLE--
B      WWWTAB.  --BRING ADDRESS OF 1ST DOPE VECTOR WORD--
AA     1U
C      WWWI1.   --STORE 1ST 4 CHARACTERS OF ITEM NAME--
AA     1U
C      WWWI2.   --STORE 2ND 4 CHARACTERS OF ITEM NAME--
AA     1U
C      WWWIE    --STORE ENTRY ORDER--
AA     1U
C      WWWDP1   --STORE 1ST TARGET DOPE VECTOR WORD--
AA     1U
C      WWWDP2   --STORE 2ND TARGET DOPE VECTOR WORD--
AA     1U
C      WWWRT    --STORE RETURN ADDRESS OF SUBROUTINE--
BE     WWWTAB.
C      WWWTAB.
AA     1U
C      WWWTD2
C      WWWA3.   --STORE ADDRESS ITEM DESCRIPTION ZONE -1--
BE     WWWIE
C      WWWIE
BE     WWWDP1
C      WWWDP1
BE     WWWDP2
C      WWWDP2
BE     WWWTD2
ET     'FFF000'H
SH     11
C      WWWB1.   --(FORMULA. ITEM NUMBER * 3 ,WITH--
SH     1        --SHIFTS FOR TIME ECONOMY)--
A      WWWB1.   --TO CALCULATE THE ADDRESS OF THE LAST--
A      WWWA3.   --DOPE VECTOR WORD OF ITEM DESCRIPTION--
C      WWWB1.   --STORAGE OF THIS ADDRESS--
WWWTA1 = B      WWWA3.   --TEST FOR ITEM EXISTENCE--
SB      WWWB1.   --(IF THE ITEM DOPE VECTOR ARE DEFINED)--
SK      5L
SU      WWWM3.   --IF ITEM DOES NOT EXIST,ERROR MESSAGE--
AAE     WWWI1.   --WITH AN ERROR SUBROUTINE--
AAE     WWWI2.   --AND SUPPLYING WORDS--
AAE     WWWTAB.  --(WITH THE ITEM NAME AND TABLE NAME)--
B      WWWA3.   --BRING ADDRESS ITEM DECRPTION ZONE -1--
AA     1U
C      WWWA1.   --STORE ADDRESS 1ST ITEM DOPE VECTOR--
AA     1U
C      WWWA2.   --STORE ADDRESS 2ND ITEM DOPE VECTOR--
AA     1U
C      WWWA3.   --STORE ADDRESS 3RD ITEM DOPE VECTOR--
BE     WWWI1.   --BRING 1ST 4 CHARACTERS OF ITEM NAME--
SBE     WWWA1.  --TEST IF THE FIRST PARTS OF ITEM NAME--
SI      2L      --ARE THE SAME--
S      WWWTA1   --IF NO,BRING NEW ITEM NAME OR ERROR--
BE     WWWA2.   --BRING AND EXTRACT THE 2ND 4 CHARACTERS--
LT      'FFF000'H --OF ITEM NAME--
SBE     WWWI2.  --TEST IF THE SECOND PARTS OF ITEM NAME--
SI      2L      --ARE THE SAME--
S      WWWTA1   --IF NO,BRING NEW ITEM NAME OR ERROR--
BE     WWWA3.   --BRING ADDRESS ITEM BEGIN IN 1ST ENTRY--
CE      WWWDP1  --STORAGE OF THIS ADDRESS--
BE     WWWIE    --CALCULATION WORD NUMBER DISPLACEMENT--
SA      1U      --TO HAVE THE RELATIVE ADDRESS OF ITEM--
CE      WWWIE    --(FORM.(ENTRY ORDER-1)*WORDS PER ENTRY)--
BE     WWWTD2
ET      'FFF'H
MLE     WWWIE
SH      LZ      23
AAE     WWWTAB.
AE      WWWDP1   --CALCULATION OF ITEM ABSOLUTE ADDRESS--
CE      WWWDP1   --STORE THIS ADDRESS IN 1ST DOPE VECTOR--
ET      'FF0000'H --TYPE AND BIT OFFSET (MUST BE 0)--
SI      6L
BE     WWWA2.   --BRING AND EXTRACT THE WORD NUMBER--
ET      '000FFF'H --OF THE ITEM,TO FILL THE 2ND DOPE VECT.--
SH      KL      12 --LEFT JUSTIFICATION OF THIS WORD NUMBER--

```

```

CE      WWWDP2      --*STORAGE OF THE 2ND DOPE VECTOR WORD*--
SE      WWWRT       --*RETURN OF THE SUBROUTINE*--
BE      WWA2.       --ARITHMETIC ARRAY CASE,BRING AND--
ET      '000FFF'H   --EXTRACT THE ITEM WORD NUMBER--
SH      KL      12   --TO FILL THE 2ND DOPE VECTOR WORD--
AT      1           --(FOR THIS CASE,2ND DIMENSION IS 1)--
CE      WWWDP2      --*STORAGE OF THE 2ND DOPE VECTOR WORD*--
SE      WWWRT       --*RETURN OF THE SUBROUTINE*--
          --FOR MACRO BCHAR--
.OK1    = CAB      ($WBCHAR NE 1) .END
WWWBCH.= 0         --SUBROUTINE FOR PROCESSING,MACRO BCHAR--
          B        WWA0.      --BRING 1ST DOPE VECTOR WORD--
          ET      'FFFF'H     --EXTRACT THE ADDRESS OF THE STRING--
          SH      L      3     --MULTIPLICATION BY 24 TO HAVE THE--
          C        WWA3.      --ADDRESS IN BITS OF FIRST WORD OF--
          SH      L      1     --THE STRING--
          A        WWA3.
          C        WWA3.
          B        WWA0.      --BRING 1ST DOPE VECTOR WORD--
          ET      '1F0000'H    --EXTRACT BIT OFFSET SPECIFICATION--
          SH      '16         --(RIGHT JUSTIFICATION)--
          A        WWA3.      --ADDRESS IN BITS OF THE STRING--
          C        WWA3.
          B        WWA0.      --BRING 1ST DOPE VECTOR WORD--
          ET      '600000'H    --EXTRACT TYPE SPECIFICATION CODE--
          AT      '200000'H    --TEST FOR 6 BIT CHARACTER CASES--
          SI      WWWBC6
          --IF CHECK,TEST FOR CORRECT TYPE SPECIFICATION--
          CAB      ($WCHECK NE 1) .OK6
          AT      '400000'H    --TEST FOR 8 BIT CHARACTER CASES--
          SI      WWWBC6-2
          SU      WWW3.      --IF NOT 6 OR 8 BIT CHARACTERS,ERROR--
          S        WWWBC6     --6 BIT CHARACTER CASE IS ASSUMED--
          .OK6    = B        8      --8 BIT CHARACTER CASE,BRING 8--
          S        WWWBC6+1
          WWWBC6 = B        6      --6 BIT CHARACTER CASE,BRING 6--
          C        WWA4.      --STORAGE OF BITS PER CHARACTER--
          B        WWA1.      --BRING CHARACTER ORDER--
          SA      1U
          ML      WWA4.      --CALCULATION OF BIT DISPLACEMENT--
          SH      LZ      23   --FOR THE FIRST CHARACTER TO BRING--
          A        WWA3.      --TO HAVE THE ABSOLUTE ADDRESS IN BITS--
          C        WWA3.      --OF THIS CHARACTER--
          ML      '2AAAAA'H    --DIVISION PER 24 TO HAVE THE ADDRESS--
          SH      3          --OF THE WORD CONTAINING THE 1ST--
          C        WWA5.      --CHARACTER TO BRING--
          AA      1U
          C        WWA6.      --ADDRESS OF 2ND WORD (TO HAVE N CHAR.) --
          B        WWA5.      --CALCULATION OF BIT OFFSET OF 1ST CHAR.--
          SH      L      3     --(MULTIPLICATION BY 24)
          C        WWA7.
          SH      L      1
          A        WWA7.
          SB      WWA3.      --TO HAVE THE BIT NUMBER TO TAKE--
          AA      24U        --IN THE FIRST WORD--
          AT      '8000'H     --AND THEREFORE,THE SHIFT NUMBER--
          C        WWA7.      --WITH THE SPECIFICATION Z --
          B        WWA2.      --BRING NUMBER OF CHARACTERS TO BRING--
          ML      WWA4.      --BIT NUMBER OF THE N CHARACTERS--
          SH      LZ      23   --TO BRING--
          AT      '8800'H     --AND THEREFORE,THE SHIFT NUMBER--
          C        WWA8.      --WITH THE SPECIFICATION ZL --
          BE      WWA6.      --BRING 2ND PART OF THE N CHARACTERS--
          SH      V      0     --IN REGISTER Q--
          BE      WWA5.      --BRING 1ST PART OF THE N CHARACTERS--
          SHE     WWA7.      --LEFT JUSTIFICATION IN REGISTER Q--
          B        'AAAAAA'H  --FILLING OF MISSING CHAR. WITH BLANKS--
          SHE     WWA8.      --RIGHT JUSTIFICATION IN ACCUMULATOR--
          SE      WWWBCH.     --RETURN OF THE SUBROUTINE--
.END    = MEND

```

```

MACRO
WEND2
  --DECLARATION OF INTERNAL VARIABLES--
  --GLOBAL VARIABLES--
DCGA  WBOOL,WIFB,WTESTB
  --TEST FOR SUBROUTINE GENERATION--
  --FOR MACRO BOOL--

```

```

CAB      ($$WBOOL NE 1) .OK2
WWWBOO.= 0      --SUBROUTINE FOR PROCESSING,MACRO BOOL--
B        24      --TEST FOR PROCESSING OF MACRO BOOL--
SB       WWWC5.  --ON MORE THAN A WORD--
SK       WWWB01  --IF YES,JUMP TO AN OTHER PROCESSING--
*BBIT    WWWC1.,WWWC2.,WWWC5.
C        WWWB0.  --BRING BITS OF 1ST STRING AND STORAGE--
*BBIT    WWWC3.,WWWC4.,WWWC5.
C        WWWB1.  --BRING BITS OF 2ND STRING AND STORAGE--
B        WWWC0.  --TEST FOR OPERATOR CODE AND RIGHT--
SK       5L      --PROCESSING (AND)--
SI       7L      --(OR)
B        WWWB0.  --XOR PROCESSING,BRING 1ST BIT STRING--
AT       WWWB1.  --AND MAKE A EXCLUSIVE OR WITH THE 2ND--
S        10L     --JUMP TO STORAGE PROCESSING--
B        WWWB0.  --AND PROCESSING,BRING 1ST BIT STRING--
ET       WWWB1.  --AND MAKE AN AND WITH THE 2ND--
S        7L      --JUMP TO STORAGE PROCESSING--
B        WWWB0.  --OR PROCESSING,BRING 1ST BIT STRING--
AT       WWWB1.  --AND MAKE A INCLUSIVE OR WITH THE 2ND--
C        WWWB2.  --(FORMULA. A EXCLUSIVE OR AND AN AND--
B        WWWB0.  --ON THE RESULT)--
ET       WWWB1.
AT       WWWB2.  --END LOGICAL PROCESSING,STORAGE PROCES.--
*CBIT    WWWC1.,WWWC2.,WWWC5.
SE       WWWB00. --END OF PROCESSING,RETURN JUMP OF SUB.--
WWWB01 = SU     WWWMB1. --BITS ON MORE THAN 1 WORD,BRING PROCES.--
B        WWWZ2.  --STORAGE OF THE ADDRESS OF THE--
C        WWWZ0N. --AUXILIARY MEMORY (2ND MEMORY)--
B        WWWC1.  --CHANGE ADDRESS OF 2ND STRING NAME--
C        WWWC3.  --WITH ADDRESS OF 1ST STRING NAME--
B        WWWC2.  --CHANGE BIT ORDER OF 2ND STRING NAME--
C        WWWC4.  --WITH BIT ORDER OF 1ST STRING NAME--
SU       WWWMB1. --BRING PROCESSING FOR 1ST STRING--
B        WWWZ1.  --STORAGE OF THE ADDRESS OF THE--
C        WWWZ0N. --AUXILIARY MEMORY (1ST MEMORY)--
C        WWWC8.
B        WWWZ2.  --(AND ALSO 2ND MEMORY)--
C        WWWC9.
B        WWWC5.  --STORAGE OF THE LENGTH--
C        WWWC7.
WWWB02 = B      WWWC0.  --BRING THE LOGICAL OPERAND AND--
SK       5L      --TEST THE LOGICAL OPERATION TO DO--
SI       7L
BE       WWWC8.  --XOR PROCESSING,BETWEEN A WORD OF 1ST--
ATC      WWWC9.  --STRING AND THE EQUIVALENT OF THE 2ND--
S        10L     --JUMP TO STORAGE OF THE RESULT--
BE       WWWC8.  --AND PROCESSING,BETWEEN A WORD OF 1ST--
ETE      WWWC9.  --STRING AND THE EQUIVALENT OF THE 2ND--
S        7L      --JUMP TO STORAGE OF THE RESULT--
BE       WWWC8.  --OR PROCESSING,BETWEEN A WORD OF 1ST--
ATE      WWWC9.  --STRING AND THE EQUIVALENT OF THE 2ND--
C        WWWB2.  --(FORMULA. A EXCLUSIVE OR AND AN AND--
BE       WWWC8.  --ON THE RESULT)--
ETE      WWWC9.
AT       WWWB2.
CE       WWWC8.  --STORAGE OF RESULT OF LOGICAL PROCES.--
B        WWWC7.  --TEST FOR END OF LOGICAL PROCESSING--
SA       24U
SK       WWWB03  --IF END OF LOGICAL PROCESSING--
SI       WWWB03  --JUMP TO STORAGE PROCESSING--
C        WWWC7.  --IF NOT END OF LOGICAL PROCESSING--
B        WWWC8.  --UPDATING OF INTERNAL WORK VARIABLES--
AA       1U
C        WWWC8.
B        WWWC9.
AA       1U
C        WWWC9.
S        WWWB02  --AND RETURN JUMP TO LOGICAL PROCESSING--
WWWB03 = SU     WWWMB3. --SUBROUTINE CALL FOR STORAGE PROCESSING--
SE       WWWB00. --RETURN JUMP OF BOOL SUBROUTINE--
--FOR MACRO IFB--
.OK2     = CAB   ($$WIFB NE 1) .OK3
WWWIFB.= 0      --SUBROUTINE FOR PROCESSING,MACRO IFB--
B        24      --TEST IF THE LENGTH IS GREATER THAN--
C        WWWA2.  --A WORD,AND STORAGE OF THE LENGTH OF--
WWWIFB1 = SB     WWWC5.  --A WORD (24 BITS) FOR THE CASE THAT--
SK       3L      --THE LENGTH IS GREATER THAN 24--
B        WWWC5.  --IF NO,STORAGE OF THE LENGTH IN AN--
C        WWWA2.  --INTERNAL WORK VARIABLE--

```

```

                                -P11-27-
B      WWWC4.      --UPDATING OF INTERNAL WORK VARIABLES--
C      WWWA1.      --FOR SUBROUTINE CALL FOR BRING PROCES.--
B      WWWC3.      --(FOR 2ND STRING)--
C      WWWA0.
SU     WWWBBI.     --SUBROUTINE CALL FOR BRING PROCESSING--
C      WWWB0.     --STORAGE IN AN INTERNAL WORK VARIABLE--
B      WWWC2.     --UPDATING OF INTERNAL WORK VARIABLES--
C      WWWA1.     --FOR SUBROUTINE CALL FOR BRING PROCES--
B      WWWC1.     --(FOR 1ST STRING)--
C      WWWA0.
SU     WWWBBI.     --SUBROUTINE CALL FOR BRING PROCESSING--
C      WWWB1.     --STORAGE IN AN INTERNAL WORK VARIABLE--
B      WWWC0.     --LOGICAL OPERAND TEST--
SK     8L
B      WWWB0.     --EQUAL PROCESSING,TEST IF THE BITS ARE--
SB     WWWB1.     --EQUAL--
SI     WWWIB2     --IF YES,JUMP TO TEST FOR END OF PROCES--
B      WWWIFB.     --IF NO,JUMP TO THE 2ND LABEL--
AA     1U
C      WWWIFB.
SE     WWWIFB.
B      WWWB0.     --NOT EQUAL PROCESSING,TEST IF THE--
SB     WWWB1.     --BITS ARE NOT EQUAL--
SI     -6L        --IF NO,JUMP TO THE 2ND LABEL--
WWWIB2 = B      WWWC5.     --IF YES,TEST FOR END OF PROCESSING--
SA     24U
SIE    WWWIFB.     --IF END OF PROCESSING--
SKE    WWWIFB.     --JUMP TO 1ST LABEL,CONDITION SATISFIED--
C      WWWC5.     --IF NOT END OF PROCESSING--
B      WWWC2.     --UPDATING OF INTERNAL WORK VARIABLES--
AA     24U
C      WWWC2.
B      WWWC4.
AA     24U
C      WWWC4.
B      24
S      WWWIB1     --AND RETURN JUMP TO BRING PROCESSING--
                                --FOR MACRO TESTB--
.OK3   = CAB     ($$WTESTB NE 1) .END
WWWTES.= 0      --SUBROUTINE FOR PROCESSING,MACRO TESTB--
B      WWWA2.     --STORAGE OF THE LENGTH AND--
C      WWWC2.     --TEST IF THIS LENGTH IS GREATER 24--
WWWTE1 = SA     24U     --IF NO,JUMP TO BRING PROCESSING--
SK     4L
SI     3L
B      24         --IF YES,STORAGE OF THE LENGTH OF A WORD--
C      WWWA2.     --FOR WORD PER WORD PROCESSING--
SU     WWWBBI.     --SUBROUTINE CALL FOR BRING PROCESSING--
SI     5L         --IF THE BITS = 0,JUMP TO END OF PROCES --
B      WWWTES.     --IF THE BITS ARE NOT EQUAL ZERO--
AA     1U         --JUMP TO THE 2ND LABEL--
C      WWWTES.     --THE CONDITION IS NOT SATISFIED--
SE     WWWTES.
B      WWWC2.     --IF THE BITS ARE EQUAL ZERO--
SA     24U         --TEST FOR END OF PROCESSING--
SIE    WWWTES.     --IF YES,JUMP TO 1ST LABEL--
SKE    WWWTES.     --THE CONDITION IS SATISFIED--
C      WWWC2.     --IF NO,UPDATING OF INTERNAL WORK VAR.--
B      WWWA1.
AA     24U
C      WWWA1.
B      WWWC2.
C      WWWA2.
S      WWWTE1     --AND RETURN JUMP TO PROCESSING--
.END   = MEND

```

```

MACRO
WEND3

```

```

--DECLARATION OF INTERNAL VARIABLES--
--GLOBAL VARIABLE--

```

```

DCGA  WMOV C
--TEST FOR SUBROUTINE GENERATION--
--FOR MACRO MOV C--

```

```

CAB   ($$WMOV C NE 1) .END
WWWMVC.= 0      --SUBROUTINE FOR PROCESSING (BEGIN) --
B      WWWC3.     --BRING DOPE VECTOR OF 2ND STRING--
ET     '600000'H  --EXTRACT THE TYPE SPECIFICATION CODE--
AT     '600000'H  --TEST FOR 6 OR 8 BIT CHARACTERS--

```

```

SI      2L      --IF 8 BIT CHARACTERS,BRING 3--
B       1       --IF 6 BIT CHARACTERS,BRING 4--
AA      3U      --(THE CHARACTER NUMBER PER WORD)--
C       WWWC6.  --IN AN INTERNAL WORK VARIABLE--
SB      WWWC5.  --TEST IF THE LENGTH IS GREATER THAN--
SK      WWWMC1. --THE CHARACTER NUMBER PER WORD--
*BCCHAR WWWC3.,WWWC4.,WWWC5.
*CCHAR  WWWC1.,WWWC2.,WWWC5.
SE      WWWMCV. --IF NO,MOVING AND RETURN JUMP--
WWWMC1.= B      WWWC5. --IF YES,STORAGE OF THE LENGTH--
C       WWWC7.  --IN AN INTERNAL WORK VARIABLE--
B       WWWZON. --BRING POINTER OF INTERNAL WORK AREA--
C       WWWC8.  --AND STORAGE,THEN BRING THE CHARACTERS--
*BCCHAR WWWC3.,WWWC4.,WWWC6.
CE      WWWC8.  --STORE THIS CHAR. IN INTERNAL WORK AREA--
B       WWWA7.  --BRING AND CONSTRUCT THE RIGHT SHIFT--
SB      '8018'H --(SH 2L BIT OFFSET) OUT OF THE MACRO--
AT      'FF77FF'H --BCHAR (WITH SH 2 (24-BIT OFF.) AT 1ST--
C       WWWA7.  --STORAGE IN AN INTERNAL WORK VARIABLE--
S       6L      --FOR 1ST 4 CHAR. JUMP TO END OF BRING --
WWWMC2.= BE      WWWA6. --REGENERATE THE 4 CHARACTERS IN A WORD--
SH      V      0 --TO STORE THIS 4 CHARACTERS IN JUST--
BE      WWWA5.  --A WORD OF THE INTERNAL WORK AREA--
SHE     WWWA7.
CE      WWWC8.
B       WWWC7.  --TEST IF THE BRING PROCESSING IS ENDED--
SB      WWWC6.
SI      WWWMC3. --IF YES,JUMP TO STORE PROCESSING--
SK      WWWMC3.
C       WWWC7.  --IF NO VARIABLE REGENERATION,RESTART--
B       WWWC6.  --INCREMENTATION OF THE ADDRESS OF THE--
C       WWWA5.  --WORK VARIABLES TO TAKE THE NEXT CHAR.--
AA      1U
C       WWWA6.
B       WWWC8.  --INCREMENTATION OF THE ADDRESS OF THE--
AA      1U      --NEXT WORD IN THE INTERNAL WORK AREA--
C       WWWC8.
S       WWWMC2. --RETURN JUMP FOR BRING PROCESSING--
WWWMC3.= B      WWWC5. --STORAGE OF THE LENGTH--
C       WWWC7.  --IN AN INTERNAL WORK VARIABLE--
B       WWWZON. --BRING POINTER OF INTERNAL WORK AREA--
C       WWWC8.  --AND STORAGE--
BE      WWWC8.  --BRING 1ST CHARS IN ACCUM. AND STORAGE--
*CCHAR  WWWC1.,WWWC2.,WWWC6.
B       WWWA7.  --BRING BIT OFFSET SPECIFICATION--
ET      '3F'H   --(24-BIT OFFSET EXACTLY)--
SA      24U     --AND TEST IF WE HAVE A BIT OFFSET--
SI      WWWMC6. --IF NO BIT OFFSET,NEXT CHAR. IN 1 WORD--
S       WWWMC5.
WWWMC4.= BE      WWWC8. --BRING NEXT CHARACTERS--
SH      V      0 --IN Q REGISTER--
BE      WWWA5.  --BRING 1ST WORD TO FILL IT WITH THE--
SHE     WWWA7.  --1ST PART OF THE NEW CHARACTERS--
SHE     WWWA10. --(RIGHT JUSTIFIED)--
CE      WWWA5.  --AND STORAGE--
BE      WWWA6.  --BRING 2ND WORD TO FILL IT WITH THE--
SHE     WWWA11. --2ND PART OF THE NEW CHARACTERS--
SHE     WWWA9.  --(LEFT JUSTIFIED)--
CE      WWWA6.  --AND STORAGE--
WWWMC5.= B      WWWC7.  --THE NEXT CHARACTERS ARE IN 2 WORDS--
SB      WWWC6.  --TEST FOR END OF PROCESSING--
SIE     WWWMCV. --(LENGTH - CHAR. NUMBER STORED = 0)--
C       WWWC7.  --IF NO,DECREMENTATION OF THE LENGTH--
B       WWWC8.  --INCREMENTATION OF THE ADDRESS OF THE--
AA      1U      --NEXT WORD IN THE INTERNAL WORK AREA--
C       WWWC8.
B       WWWA6.  --INCREMENTATION OF THE ADDRESS OF THE--
C       WWWA5.  --NEXT CHARACTERS,WHICH ARE IN 2 WORDS--
AA      1U
C       WWWA6.
B       WWWC7.  --TEST FOR LAST CHARACTERS PROCESSING--
SB      WWWC6.  --(FOR 1 TO 2 OR 3 LAST CHARACTERS)--
SK      2L
S       WWWMC4. --IF NO LAST CHAR. JUMP TO STORAGE--
BE      WWWC8.  --SPECIAL PROCESSING FOR LAST CHARACTERS--
C       WWWB0.  --STORAGE OF THIS CHARACTERS--
B       WWWMCV. --BRING ADDRESS OF RETURN JUMP IN AN--
C       WWWCCH. --INTERNAL WORK VARIABLE OF MACRO CCHAR--
B       WWWC7.  --BRING NUMBER OF LAST CHARACTERS--
S       WWWCCM. --AND JUMP IN MACRO CCHAR FOR PROCESSING--

```

```

WWVMC6.= B      WWWC7.      --THE NEXT CHARACTERS ARE IN 1 WORD--
          SB      WWWC6.      --TEST FOR END OF PROCESSING--
          SIE     WWWMVC.      --(LENGTH - CHAR. NUMBER STORED = 0)--
          C       WWWC7.      --IF NO, DECREMENTATION OF THE LENGTH--
          B       WWWC8.      --INCREMENTATION OF THE ADDRESS OF THE--
          AA      1U          --NEXT IN THE INTERNAL WORK AREA--
          C       WWWC8.
          B       WWWA5.      --INCREMENTATION OF THE ADDRESS--
          AA      1U          --OF THE NEXT CHARACTERS--
          C       WWWA5.
          B       WWWC7.      --TEST FOR LAST CHARACTERS PROCESSING--
          SB      WWWC6.      --(FOR 1 TO 2 OR 3 LAST CHARACTERS)--
          SK      4L
          BE      WWWC8.      --IF NO LAST CHAR. BRING THE CHARACTERS--
          CE      WWWC5.      --AND STORAGE--
          S       WWWMC6.      --JUMP FOR NEXT CHAR. PROCESSING--
          B       WWWC7.      --LAST CHARACTERS PROCESSING, BRING--
          ML      WWWA4.      --BIT NUMBER TO TRANSFER--
          SH      LZ 23
          AT      '8800'H      --CONSTRUCTION OF SHIFTS FOR FORMATTING--
          C       WWWC7.      --OF THE LAST WORD--
          AT      'C800'H      --(WITH KEEPING OF REMAINING BITS)--
          C       WWWC9.
          BE      WWWC8.      --BRING NEW CHARACTERS IN Q REGISTER--
          SH      V 0
          BE      WWWA5.      --BRING OLD CHARACTERS IN ACCUMULATOR--
          SHE     WWWC7.      --EXTRACT THE RIGHT REMAINING CHARACT.--
          SHE     WWWC9.      --AND PLACE THE NEW CHARACTERS IN THE--
          CE      WWWA5.      --RIGHT POSITION AND STORAGE--
          SE      WWWMVC.      --RETURN JUMP OF THE SUBROUTINE--
.END      = MEND

```

MACRO
WEND4

--DECLARATION OF INTERNAL VARIABLES--
--GLOBAL VARIABLES--
DCGA WMOV8, WMB1, WMB3
--TEST FOR SUBROUTINE GENERATION--
--FOR MACRO MOV8--

```

CAB      ($$WMOV8 NE 1) .OK4
WWVMV8.= 0      --SUBROUTINE FOR PROCESSING, MACRO MOV8--
          B       24          --TEST IF THE LENGTH IS GREATER THAN--
          SB      WWWC5.      --A WORD (THE BITS ARE ON SEVERAL WORDS)--
          SK      WWWMB1
          *BBIT   WWWC3., WWWC4., WWWC5.
          *CBIT   WWWC1., WWWC2., WWWC5.
          SE      WWVMV8.      --IF NO, MOVING AND RETURN JUMP--
WWMB1 = SU     WWWMB1.
          SU      WWWMB3.
          SE      WWVMV8.      --RETURN JUMP OF THE SUBROUTINE--
          --FOR MACRO BOOL AND MOV8--
.OK4     = CAB   ($$WMB1 NE 1) .OK5
WWMB1.= 0
          B       WWWC5.      --IF YES, STORAGE OF THE LENGTH--
          C       WWWC7.      --IN AN INTERN WORK VARIABLE--
          B       WWWZON.      --BRING POINTER OF INTERNAL WORK AREA--
          C       WWWC8.      --AND STORAGE, THEN BRING 24 BITS--
          *BBIT   WWWC3., WWWC4., 24
          CE      WWWC8.      --STORE THIS 24 BITS IN INTERN WORK AREA--
          B       WWWA7.      --BRING AND CONSTRUCT THE RIGHT SHIFT--
          SB      '8018'H      --(SH ZL BIT OFFSET) OUT OF THE MACRO--
          AT      'FF77FF'H      --BBIT (SH Z (24-BIT OFFSET) AT 1ST.)--
          C       WWWA7.      --STORAGE IN AN INTERNAL WORK VARIABLE--
          S       6L          --FOR 1ST 24 BITS, JUMP TO END OF BRING --
WWMB2 = BE     WWWA6.      --REGENERATE THE 24 BITS IN A WORD--
          SH      V 0          --TO STORE THIS 24 BITS IN JUST--
          BE      WWWA5.      --A WORD OF THE INTERNAL WORK AREA--
          SHE     WWWA7.
          CE      WWWC8.
          B       WWWC7.      --TEST IF THE BRING PROCESSING IS ENDED--
          SA      24U
          SI      WWWMB3      --IF YES, JUMP TO STORE PROCESSING--
          SK      WWWMB3
          C       WWWC7.      --IF NO, VARIABLE REGENERATION, RESTART--
          B       WWWA6.      --INCREMENTATION OF THE ADDRESS OF THE--
          C       WWWA5.      --WORK VARIABLES TO TAKE THE NEXT BITS--
          AA      1U
          C       WWWA6.

```

```

      B   WWWC8.      --INCREMENTATION OF THE ADDRESS OF THE--
      AA  1U         --NEXT WORD IN THE INTERNAL WORK AREA--
      C   WWWC8.
      S   WWWMB2.    --RETURN JUMP FOR BRING PROCESSING--
WWWMB3 = SE   WWWMB1.
.OK5  = CAB  ($$WWWB3 NE 1) .END
WWWMB3.= 0
      B   WWWC5.      --STORAGE OF THE LENGTH--
      C   WWWC7.      --IN AN INTERNAL WORK VARIABLE--
      B   WWWZON.     --BRING POINTER OF INTERNAL WORK AREA--
      C   WWWC8.      --AND STORAGE--
      BE  WWWC8.      --BRING 1ST. BITS IN ACCUM. AND STORAGE--
*CBIT   WWWC1.,WWW2.,24
      B   WWWA7.      --BRING BIT OFFSET SPECIFICATION--
      ET  '3F'H      --(24-BIT OFFSET EXACTLY)--
      SA  24U         --AND TEST IF WE HAVE A BIT OFFSET
      SI   WWWMB6     --IF NO BIT OFFSET,NEXT 24 BITS--
      S   WWWMB5     --IN ONE WORD--
WWWMB4 = BE  WWWC8.    --BRING NEXT 24 BITS--
      SH  V  0        --IN Q REGISTER--
      BE  WWWA5.      --BRING 1ST WORD TO FILL IT WITH THE--
      SHE WWWA7.      --1ST PART OF THE NEW BITS--
      SHE WWWA10.     --(RIGHT JUSTIFIED)--
      CE  WWWA5.      --AND STORAGE--
      BE  WWWA6.      --BRING 2ND WORD TO FILL IT WITH THE--
      SHE WWWA11.     --2ND PART OF THE NEW BITS--
      SHE WWWA9.      --(LEFT JUSTIFIED)--
      CE  WWWA6.      --AND STORAGE--
WWWMB5 = B   WWWC7.    --THE NEXT 24 BITS ARE IN 2 WORDS--
      SA  24U         --TEST FOR END OF PROCESSING--
      SIE WWWMB3.
      C   WWWC7.      --IF NO,DECREMENTATION OF THE LENGTH--
      B   WWWC8.      --INCREMENTATION OF THE ADDRESS OF THE--
      AA  1U         --NEXT WORD IN THE INTERNAL WORK AREA--
      C   WWWC8.
      B   WWWA6.      --INCREMENTATION OF THE ADDRESS OF THE--
      C   WWWA5.      --NEXT 24 BITS,WHICH ARE IN 2 WORDS--
      AA  1U
      C   WWWA6.
      B   WWWC7.      --TEST FOR LAST BITS PROCESSING--
      SA  24U         --(FOR 1 TO 23 LAST BITS)--
      SK  2L
      S   WWWMB4     --IF NO LAST BITS JUMP TO STORAGE--
      BE  WWWC8.      --SPECIAL PROCESSING FOR LAST BITS--
      C   WWWB0.      --STORAGE OF THE BITS--
      B   WWWMB3.     --BRING ADDRESS OF RETURN JUMP IN AN--
      C   WWWCB1.     --INTERNAL WORK VARIABLE OF MACRO CBIT--
      B   WWWC7.      --BRING NUMBER OF LAST BITS--
      C   WWWA2.      --STORAGE IN INT. WORK VARIABLE OF CBIT--
      B   WWWA7.
      S   WWWCBM.     --AND JUMP IN MACRO CBIT FOR PROCESSING--
WWWMB6 = B   WWWC7.    --THE NEXT 24 BITS ARE IN 1 WORD--
      SA  24U         --TEST FOR END OF PROCESSING--
      SIE WWWMB3.
      C   WWWC7.      --IF NO,DECREMENTATION OF THE LENGTH--
      B   WWWC8.      --INCREMENTATION OF THE ADDRESS OF THE--
      AA  1U         --NEXTWORD IN THE INTERNAL WORK AREA--
      C   WWWC8.
      B   WWWA5.      --INCREMENTATION OF THE ADDRESS--
      AA  1U         --OF THE NEXT 24 BITS--
      C   WWWA5.
      B   WWWC7.      --TEST FOR LAST BITS PROCESSING--
      SA  24U         --(FOR 1 TO 23 BITS)--
      SK  4L
      BE  WWWC8.      --IF NO LAST BITS,BRING THE BITS--
      CE  WWWA5.      --AND STORAGE--
      S   WWWMB6     --JUMP FOR NEXT 24 BITS PROCESSING--
      B   WWWC7.      --LAST BITS PROCESSING--
      AT  '8800'H    --CONSTRUCTION OF SHIFTS FOR FORMATTING--
      C   WWWC7.      --OF THE LAST WORD--
      AT  'C800'H    --(WITH KEEPING OF REMAINING BITS)--
      C   WWWC9.
      BE  WWWC8.      --BRING NEW BITS IN Q REGISTER--
      SH  V  0
      BE  WWWA5.      --BRING OLD BITS IN ACCUMULATOR--
      SHE WWWC7.      --EXTRACT THE RIGHT REMAINING BITS--
      SHE WWWC9.      --AND PLACE THE NEW BITS IN THE--
      CE  WWWA5.      --RIGHT POSITION AND STORAGE--
      SE  WWWMB3.
.END  = MEND

```

MACRO
WEND5

- 411-31-

--DECLARATION OF INTERNAL VARIABLES--
--GLOBAL VARIABLES--

DCGA WCCHAR,WIFC

--TEST FOR SUBROUTINE GENERATION--
--FOR MACRO CCHAR--

CAB (\$\$WCCHAR NE 1) .OK6

WWWCCM.= 0 --SUBROUTINE FOR PROCESSING,MACRO CCHAR--

B WWWWA0. --BRING 1ST DOPE VECTOR WORD--

ET 'FFFF'H --EXTRACT THE ADDRESS OF THE STRING--

SH L 3 --MULTIPLICATION BY 24 TO HAVE THE--

C WWWWA3. --ADDRESS IN BITS OF FIRST WORD OF--

SH L 1 --THE CHARACTER STRING TO STORE--

A WWWWA3.

C WWWWA3.

B WWWWA0. --BRING 1ST DOPE VECTOR WORD--

ET '1F0000'H --EXTRACT BIT OFFSET SPECIFICATION--

SH 16 --(RIGHT JUSTIFICATION)--

A WWWWA3. --ADDRESS IN BITS OF THE STRING--

C WWWWA3.

B WWWWA0. --BRING 1ST DOPE VECTOR WORD--

ET '600000'H --EXTRACT TYPE SPECIFICATION CODE--

AT '200000'H --TEST FOR 6 BIT CHARACTER CASES--

SI WWWCC6

--IF CHECK,TEST FOR CORRECT TYPE SPECIFICATION--

CAB (\$\$WCHECK NE 1) .OK5

AT '400000'H --TEST FOR 8 BIT CHARACTER CASES--

SI WWWCC6-2

SU WWWM5. --IF NOT BIT TYPE,ERROR MESSAGE--

S WWWCC6 --6 BIT CHARACTER CASE IS ASSUMED--

B 8 --8 BIT CHARACTER CASE,BRING 8--

S WWWCC6+1

.OK5 = NOP

WWWCC6 = B

6 --6 BIT CHARACTER CASE,BRING 6--

C WWWWA4. --STORAGE OF BITS PER CHARACTER--

B WWWWA1. --BRING CHARACTER ORDER--

SA 1U

ML WWWWA4. --CALCULATION OF BIT DISPLACEMENT--

SH LZ 23 --FOR THE FIRST CHARACTER TO BE STORE--

A WWWWA3. --TO HAVE THE ABSOLUTE ADDRESS IN BITS--

C WWWWA3. --OF THIS CHARACTER--

ML '2AAAAB'H --DIVISION PER 24 TO HAVE THE ADDRESS--

SH 3 --OF THE WORD CONTAINING THE 1ST--

C WWWWA5. --CHARACTER TO STORE--

AA 1U

C WWWWA6. --ADDRESS OF 2ND WORD (TO HAVE N CHAR.) --

B WWWWA5. --CALCULATION OF BIT OFFSET OF 1ST CHAR.--

SH L 3 --(MULTIPLICATION BY 24)

C WWWWA7.

SH L 1

A WWWWA7.

SB WWWWA3. --TO HAVE THE BIT NUMBER TO BE TAKEN--

AA 24U --IN THE FIRST WORD--

C WWWWA7.

B WWWWA2. --BRING NUMBER OF CHAR. TO BE STORE--

WWWCCM.= ML WWWWA4. --BIT NUMBER OF THE N CHARACTERS--

SH LZ 23 --TO STORE--

C WWWWA8.

SB WWWWA7. --TEST TO KNOW IF THE CHARACTERS ARE--

SK WWWCCS --CONTAINED IN JUST ONE WORD--

SI WWWCCJ --TEST FOR RIGHT JUSTIF. IN THE WORD--

AT '8800'H --IF THE N CHARACTERS ARE OF TWO WORDS--

C WWWWA11. --CONSTRUCTION OF SHIFT INSTRUCTIONS--

AT 'C800'H --FOR FORMATTING OF--

C WWWWA9. --THE SECOND WORD TO FILL WITH CHARACT.--

B WWWWA7. --BRING BIT NUMBER TO FILL IN 1ST WORD--

AT '8800'H --AND CONSTRUCTION OF SHIFT INSTRUCTION--

C WWWWA10. --FOR FORMATTING OF 1ST WORD TO FILL--

B WWWWA8. --CONSTRUCTION OF SHIFT INSTRUCTION--

AT '800'H --FOR PREPARATION (TRANSFER OF THE--

C WWWWA8. --CHARACTER IN Q REGISTER)--

B WWWWB0. --PROCESSING PART,BRING THE CHARACTERS--

SHE WWWWA8. --IN Q REGISTER--

BE WWWWA5. --BRING 1ST WORD TO FILL WITH THE 1ST--

SHE WWWWA7. --PART OF THE N CHARACTERS--

SHE WWWWA10. --AND FILLING OF THIS WORD--

CE WWWWA5. --*STORAGE OF THE 1ST PART OF CHARACTERS*

BE WWWWA6. --BRING 2ND WORD TO FILL WITH THE 2ND--

```

SHE   WWA11.  --PART OF THE N CHARACTERS--
SHE   WWA9.   --AND FILLING OF THIS WORD--
CE    WWA6.   --*STORAGE OF THE 2ND PART OF CHARACTERS*--
SE    WWCCH.  --*RETURN OF THE SUBROUTINE*--
WWCCS = AT   'FFB7FF'H --IF THE N CHARACTERS ARE JUST IN ONE--
C     WWA9.   --WORD, BUT NOT RIGHT JUSTIFIED--
B     WWA7.   --CONSTRUCTION OF SHIFT INSTRUCTIONS--
AT    '4000'H --TO FILL THIS WORD WITH ALL THE--
C     WWA7.   --NECESSARY DISPLACEMENTS--
B     WWA8.   --AND TO CONSERV THE REMAINING BITS--
AT    '8800'H
C     WWA10.
AT    '800'H
C     WWA8.
B     WWB0.   --PROCESSING PART, BRING THE CHARACTERS--
SHE   WWA8.   --IN Q REGISTER--
BE    WWA5.   --BRING THE WORD TO FILL WITH THE CHAR.--
SHE   WWA7.   --PLACE ALL THE BITS TO CONSERV--
SHE   WWA10.  --BRING THE CHARACTERS IN RIGHT PLACE--
SHE   WWA9.   --RIGHT JUSTIFICATION WITH THE OLD BITS--
CE    WWA5.   --*STORAGE OF THE N CHARACTERS*--
SE    WWCCH.  --*RETURN OF THE SUBROUTINE*--
WWCCJ = B     WWA7.   --IF THE N CHARACTERS ARE JUST IN ONE--
AT    '8800'H --WORD, AND RIGHT JUSTIFIED--
C     WWA10.  --CONSTRUCTION OF SHIFT INSTRUCTIONS--
AT    '800'H  --TO FILL THIS WORD WITH CONSERVATION--
C     WWA8.   --OF REMAINING BITS LEFT JUSTIFIED--
B     WWB0.   --PROCESSING PART, BRING THE CHARACTERS--
SHE   WWA8.   --IN Q REGISTER--
BE    WWA5.   --BRING THE WORD TO FILL WITH THE CHAR.--
SHE   WWA7.   --PLACE THE REMAINING BITS JUST BEFORE--
SHE   WWA10.  --THE CHAR. AND FILL THE WORD--
CE    WWA5.   --*STORAGE OF THE N CHARACTERS*--
SE    WWCCH.  --*RETURN OF THE SUBROUTINE*--
--FOR MACRO IFC
.OK6  = CAB   ($$WIFC NE 1) .END
WWWIFC.= 0    --SUBROUTINE FOR PROCESSING (BEGIN)--
B     WWC1.   --TEST FOR 6 BIT OR 8 BIT CHARACTERS--
ET    '600000'H --WITH EXTRACTION OF THE TYPE SPECIF.--
AT    '200000'H --AT FIRST AND THE TEST--
SI    2L
B     2       --FOR 8 BIT CHARACTERS, BRING 8--
AA    6U      --FOR 6 BIT CHARACTERS, BRING 6--
C     WWC6.   --FOR CALCULATING IN BITS--
B     WWC2.   --TRANSLATION OF THE CHARACTER ORDER--
SA    1U      --IN THE EQUIVALENT BIT ORDER--
ML    WWC6.   --FOR THE 1ST STRING--
SH    LZ 23
AA    1U
C     WWC2.   --STORAGE IN AN INTERNAL WORK VARIABLE--
B     WWC4.   --TRANSLATION OF THE CHARACTER ORDER--
SA    1U      --IN THE EQUIVALENT BIT ORDER--
ML    WWC6.   --FOR THE 2ND STRING--
SH    LZ 23
AA    1U
C     WWC4.   --TRANSLATION OF THE LENGTH IN CHARACTER--
ML    WWC6.   --IN AN EQUIVALENT LENGTH IN BITS--
SH    LZ 23
C     WWC5.   --STORAGE IN AN INTERNAL WORK VARIABLE--
B     WWCIFC. --SUBROUTINE CALL FOR THE ALL PROCESSING--
C     WWCIFB. --(SIMULATED, WITH UPDATING OF ADDRESS)--
S     WWCIFB.+1 --JUMP TO THIS PROCESSING SUBROUTINE--
.END   = MEND

```

MACRO
WEND6

--DECLARATION OF INTERNAL VARIABLES--

--GLOBAL VARIABLES--

DGGA WBBIT, WCBIT, WMOVC, WMOVB, WBOOL

--TEST FOR SUBROUTINE GENERATION--

--FOR MACRO BBIT--

CAB (\$\$WBBIT NE 1) .OK7

WWWBBI.= 0 --SUBROUTINE FOR PROCESSING, MACRO BBIT--

--IF CHECK, TEST FOR CORRECT TYPE SPECIFICATION--

CAB (\$\$WCHECK NE 1) .OK6

B WWA0. --BRING 1ST DOPE VECTOR WORD--

ET '600000'H --EXTRACT TYPE SPECIFICATION CODE--

AT '400000'H --TEST FOR BIT SPECIFICATION CODE--

```

      SI      2L
      SU      WWWW5.    --IF NOT BIT TYPE,ERROR MESSAGE--
.OK6  = B      WWWWAO.  --BRING 1ST DOPE VECTOR WORD--
      ET      'FFFF'H  --EXTRACT THE ADDRESS OF THE STRING--
      SH      L        3 --MULTIPLICATION BY 24 TO HAVE THE--
      C      WWWW3.    --ADDRESS IN BITS OF FIRST WORD OF--
      SH      L        1 --THE BIT STRING--
      A      WWWW3.    --(WITH SHIFTS FOR TIME ECONOMY)--
      C      WWWW3.
      B      WWWWAO.    --BRING 1ST DOPE VECTOR WORD--
      ET      '1F0000'H --EXTRACT BIT OFFSET SPECIFICATION--
      SH      16       --(RIGHT JUSTIFICATION)--
      A      WWWW3.    --ADDRESS IN BITS OF THE STRING--
      A      WWWW1.    --ADDRESS OF THE FIRST BIT TO BRING--
      SA      1U       --(IN BITS)
      C      WWWW3.
      ML      '2AAAA8'H --DIVISION PER 24 TO HAVE THE ADDRESS--
      SH      3        --OF THE WORD CONTAINING THE 1ST--
      C      WWWW5.    --BIT TO BRING--
      AA      1U
      C      WWWW6.    --ADDRESS OF 2ND WORD (TO HAVE N BITS) --
      SA      1U       --CALCULATION OF BIT OFFSET OF 1ST BIT--
      SH      L        3 --(MULTIPLICATION BY 24)
      C      WWWW7.
      SH      L        1
      A      WWWW7.
      SB      WWWW3.    --TO HAVE THE BIT NUMBER TO TAKE--
      AA      24U      --IN THE FIRST WORD--
      AT      '8000'H  --AND THEREFORE,THE SHIFT NUMBER--
      C      WWWW7.    --WITH THE SPECIFICATION 2 --
      B      WWWW2.    --BRING NUMBER OF BITS TO BRING--
      AT      '8800'H  --AND THEREFORE,THE SHIFT NUMBER--
      C      WWWW8.    --WITH THE SPECIFICATION 2L --
      BE      WWWW6.    --BRING 2ND PART OF THE N BITS--
      SH      V        0 --IN REGISTER Q--
      BE      WWWW5.    --BRING 1ST PART OF THE N BITS--
      SHE     WWWW7.    --LEFT JUSTIFICATION IN REGISTER Q--
      B      0         --FILLING OF MISSING BITS WITH ZEROS--
      SHE     WWWW8.    --RIGHT JUSTIFICATION IN ACCUMULATOR--
      SE      WWWWB1.   --RETURN OF THE SUBROUTINE--
      --FOR MACRO CBIT--
.OK7  = CAB      ($$WCBIT NE 1) .OK8
      WWWWCB1.= 0     --SUBROUTINE FOR PROCESSING,MACRO CBIT--
      --IF CHECK,TEST FOR CORRECT TYPE SPECIFICATION--
      CAB      ($$WCHECK NE 1) .OK10
      B      WWWWAO.    --BRING 1ST DOPE VECTOR WORD--
      ET      '600000'H --EXTRACT TYPE SPECIFICATION CODE--
      AT      '400000'H --TEST FOR BIT SPECIFICATION CODE--
      SI      2L
      SU      WWWW5.    --IF NOT BIT TYPE,ERROR MESSAGE--
.OK10 = NOP
      B      WWWWAO.    --BRING 1ST DOPE VECTOR WORD--
      ET      'FFFF'H  --EXTRACT THE ADDRESS OF THE STRING--
      SH      L        3 --MULTIPLICATION BY 24 TO HAVE THE--
      C      WWWW3.    --ADDRESS IN BITS OF FIRST WORD OF--
      SH      L        1 --THE BIT STRING TO STORE--
      A      WWWW3.
      C      WWWW3.
      B      WWWWAO.    --BRING 1ST DOPE VECTOR WORD--
      ET      '1F0000'H --EXTRACT BIT OFFSET SPECIFICATION--
      SH      16       --(RIGHT JUSTIFICATION)--
      A      WWWW3.    --FOR CALCULATION OF THE ADDRESS IN BITS--
      A      WWWW1.
      SA      1U
      C      WWWW3.    --ADDRESS IN BITS OF THE STRING--
      ML      '2AAAA8'H --DIVISION PER 24 TO HAVE THE ADDRESS--
      SH      3        --OF THE WORD CONTAINING THE 1ST--
      C      WWWW5.    --BIT TO STORE--
      AA      1U
      C      WWWW6.    --ADDRESS OF 2ND WORD (TO HAVE N BIT) --
      SA      1U       --CALCULATION OF BIT OFFSET OF 1ST BIT--
      SH      L        3 --(MULTIPLICATION BY 24)
      C      WWWW7.
      SH      L        1
      A      WWWW7.
      SB      WWWW3.    --TO HAVE THE BIT NUMBER TO TAKE--
      AA      24U      --IN THE FIRST WORD--
      C      WWWW7.
      WWWWCBM = SB     WWWW2.  --TEST TO KNOW IF THE BITS ARE--
      SK      WWWWCB1.  --CONTAINED IN JUST ONE WORD--

```

```

AT      '4800'H      --IF THE N BITS ARE JUST IN ONE WORD--
C      WWA9.
B      WWA7.      --CONSTRUCTION OF SHIFT INSTRUCTIONS--
AT      '4000'H      --TO FILL THIS WORD WITH ALL THE--
C      WWA7.      --NECESSARY DISPLACEMENTS--
B      WWA2.      --AND TO CONSERV THE REMAINING BITS--
AT      '8800'H
C      WWA10.
AT      '800'H
C      WWA8.
B      WWB0.      --PROCESSING PART,BRING THE BITS--
SHE     WWA8.      --IN Q REGISTER--
BE      WWA5.      --BRING THE WORD TO FILL WITH THE BITS--
SHE     WWA7.      --PLACE ALL THE BITS TO CONSERV--
SHE     WWA10.     --BRING THE BITS IN RIGHT PLACE--
SHE     WWA9.      --RIGHT JUSTIFICATION WITH THE OLD BITS--
CE      WWA5.      --*STORAGE OF THE N BITS*--
SE      WWCBI.     --*RETURN OF THE SUBROUTINE*--
WWNCBD = AT      'FF77FF'H --IF THE N BITS ARE OF TWO WORDS--
C      WWA11.     --CONSTRUCTION OF SHIFT INSTRUCTIONS--
AT      'C800'H     --FOR FORMATTING OF--
C      WWA9.      --THE SECOND WORD TO FILL WITH BITS--
B      WWA7.      --BRING BIT NUMBER TO FILL IN 1ST WORD--
AT      '8800'H     --AND CONSTRUCTION OF SHIFT INSTRUCTIONS--
C      WWA10.     --FOR FORMATTING OF 1ST WORD TO FILL--
B      WWA2.      --CONSTRUCTION OF SHIFT INSTRUCTION--
AT      '8000'H     --FOR PREPARATION (TRANSFER OF THE BITS--
C      WWA8.      --IN Q REGISTER)--
B      WWB0.      --PROCESSING PART,BRING THE BITS--
SHE     WWA8.      --IN Q REGISTER--
BE      WWA5.      --BRING 1ST WORD TO FILL WITH THE 1ST--
SHE     WWA7.      --PART OF THE N BITS--
SHE     WWA10.     --AND FILLING OF THIS WORD--
CE      WWA5.      --*STORAGE OF THE 1ST PART OF BITS*--
BE      WWA6.      --BRING 2ND WORD TO FILL WITH THE 2ND--
SHE     WWA11.     --PART OF THE N BITS--
SHE     WWA9.      --AND FILLING OF THIS WORD--
CE      WWA6.      --*STORAGE OF THE 2ND PART OF BITS*--
SE      WWCBI.     --*RETURN OF THE SUBROUTINE*--
--SUBROUTINE FOR INITIALIZATION--

.OK8    = NOP
WWWINI.= 0          --SUBROUTINE FOR INITIALIZATION,IF WANTED--
--TESTS IF AN INITIALIZATION IS WANTED--
CAB      ($$MBOVC EQ 1) .OK9
CAB      ($$MBOVB EQ 1) .OK9
CAB      ($$WBOOL EQ 1) .OK9
SE      WWWINI.    --INITIALIZATION NOT WANTED,RETURN JUMP--
UAB      .END

--INITIALIZATION AND RESERVATION PART--

.OK9    = NOP
WWWZZ1.= (85)      --RESERVATION OF 1ST INTERMEDIATE MEMORY--
WWWZZ2.= (85)      --RESERVATION OF 2ND INTERMEDIATE MEMORY--
B      WWWZ1.      --INITIALIZATION OF AN INTERNAL WORK VAR--
C      WWWZON.     --WITH ADDRESS OF 1ST INTER. MEMORY--
SE      WWWINI.    --RETURN JUMP OF THE SUBROUTINE--
WWWZ1. = AA      WWWZZ1. --INITIALIZATION OF TWO--
WWWZ2. = AA      WWWZZ2. --INTERNAL WORK VARIABLES--
.END    = MEND
ND ,

```

```

MACRO
WEND7

```

```

--DECLARATION OF INTERNAL VARIABLES--
--GLOBAL VARIABLES--

```

```

DCGA     WHEXIN,WBININ,WHEXOU,WBINOU
CAB      ($$WHEXIN NE 1) .BININ1
WHEXIN.= 0
B      WWA4.
SA      6U
SI      WHEX1.
SK      WHEX1.
SH      V      0
C      WWA3.
*FOR     WWI1,1,1,WWA4.
SUEY     EINGAB
SA      39U
SI      -2L
AA      39U

```

```

SH      L      20
SH      ZLK    4
B        WWWW3.
SA      5U
SI      4L
AA      6U
C        WWWW3.
S      12L
SH      ZL     24
CE      WWWW1.
B        WWWW1.
AA      1U
C        WWWW1.
SH      V      0
C        WWWW3.
B        WWWW4.
SB      WWWWI
SA      5U
SK      3L
*END
SE      WHEXIN.
AT      'FFFFFF'H
C        WWWW4.
WHEX1. = *FOR WWWWI,1,1,WWW4.
SUEY    EINGAB
SA      39U
SI      -2L
AA      39U
SH      L      20
SH      ZLK    4
*END
SH      ZL     24
CE      WWWW1.
B        6
SB      WWWW4.
SH      L      2
AT      '800'H
C        WWWW2.
BE      WWWW1.
SHE     WWWW2.
CE      WWWW1.
SE      WHEXIN.
.BININ1= CAB   ($$WBININ NE 1) .HEXOU1
WBININ.= 0
B        WWWW4.
SA      24U
SI      WBINI.
SK      WBINI.
SH      V      0
C        WWWW3.
*FOR WWWWI,1,1,WWW4.
SUEY    EINGAB
SI      4L
SA      39U
S      -3L
AA      39U
SH      Z      1
B        WWWW3.
SA      23U
SI      4L
AA      24U
C        WWWW3.
S      13L
SH      ZL     24
SH      V      24
CE      WWWW1.
B        WWWW1.
AA      1U
C        WWWW1.
SH      V      0
C        WWWW3.
B        WWWW4.
SB      WWWWI
SA      23U
SK      3L
*END
SE      WBININ.
AT      'FFFFFF'H
C        WWWW4.
WBINI. = *FOR WWWWI,1,1,WWW4.

```

```

SUEY  EINGAB
SI     4L
SA     39U
SI     -3L
AA     39U
SH     Z    1
*END
SH     ZL    24
SH     V    24
CE     WWWW1.
B      24
SB     WWWW4.
AT     '800'H
C      WWWW3.
BE     WWWW1.
SHE    WWWW3.
CE     WWWW1.
SE     WBININ.
.HEXOU1= CAB ($$WHEXOU NE 1) .BINOU1
WHEXOU.= 0
B      WWWW4.
SA     6U
SI     WHEXO.
SK     WHEXO.
*FOR   WWWI,1,6,WWW4.
BE     WWWW1.
SUEY   AUSSED
B      WWWW1.
AA     1U
C      WWWW1.
B      WWWW4.
SB     WWWI
SA     5U
SK     3L
*END
SE     WHEXOU.
AT     'FFFFFF'H
C      WWWW4.
WHEXO. = BE     WWWW1.
C      WWWW5.
*FOR   WWWI,1,1,WWW4.
B      WWWW5.
SH     V    0
SH     ZL    4
C      WWWW2.
SH     ZL    24
C      WWWW5.
B      WWWW2.
SUEY   AUSGAB
*END
SE     WHEXOU.
.BINOU1= CAB ($$WBINOU NE 1) .ERRMES
WBINOU.= 0
B      WWWW4.
SA     24U
SI     WBINO.
SK     WBINO.
SH     V    0
C      WWWW3.
BE     WWWW1.
C      WWWW5.
*FOR   WWWI,1,1,WWW4.
B      WWWW5.
SH     V    0
SH     ZL    1
C      WWWW2.
SH     ZL    24
C      WWWW5.
B      WWWW2.
SUEY   AUSGAB
B      WWWW3.
SA     23U
SI     4L
AA     24U
C      WWWW3.
S      12L
SH     V    0
C      WWWW3.
B      WWWW1.
AA     1U

```

ANNEXE 3

MACRO-CORAL

Description sommaire des macros qui composent le langage
MACRO-CORAL.

Les noms entre parenthèses, sont des mots clefs pour le langage.
Ils représentent les différentes valeurs effectives qui peuvent avoir les
paramètres correspondants.

Un astérisque précédant le nom, indique un paramètre facultatif.

1 - BEGIN

Macro d'initialisation, qui doit précéder toutes les autres.

PARAMETRES :

PRONOM	Nom du programme
*PAR1	Paramètre facultatif à être utilisé uniquement pendant la période de test (PAR1 = CHECK)

2 - D C L

Macro de déclaration

PARAMETRES

NAME	Nom de l'élément à déclarer
TYPE	Paramètre indicatif du type de l'élément (TYPE = WORD, BIT, CHAR, CHAR8)
DIM1	Première dimension
DIM2	Deuxième dimension
INIT	Paramètre d'initialisation (INIT = INIT)
ALIAS	Adresse où se trouvent les valeurs initiales

PARAMETRES :

NAME Nom de la chaîne
 IELEM Numero d'ordre du caractère (bits)
 N Nombre des caractères (bits) à chercher

10 - CCHAR et CBIT

Macros utilisées pour stocker de 1 à 4 (24) caractères (bits) présents dans l'accumulateur, dans une chaîne donnée.

PARAMETRES :

NAME Nom de la chaîne
 IELEM Numéro d'ordre du caractère (bit) dans l'accumulateur
 +N Nombre de caractère (bits) à stocker

11 - MOVC et MOVB

Macros utilisées pour le transfert de chaînes ou de sous-chaînes de caractères (bits).

PARAMETRES :

STR1 Chaîne source
 I1 Ordre du premier élément à transférer dans la chaîne source
 STR2 Chaîne objet
 I2 Position dans la chaîne objet où sera placé le premier caractère (bit) transféré
 LENGTH Nombre de caractères (bits) à transférer.

12 - BOOL

Macro utilisée pour effectuer des opérations booléennes.

PARAMETRES :

STR1 Premier opérande, qui contiendra à la fin, le résultat de l'opération
 I1 Ordre du premier bit dans STR1, qui sera utilisé pour l'opération
 STR2 Deuxième opérande
 I2 Il s'applique pour STR2, de la même façon que I1 pour STR1
 LENGTH Nombre de bits qui interviennent dans l'opération
 OPER Opérateur (OPER = AND, OR, XOR)

13 - FOR - END

Ces deux macros déterminent une boucle au moment de l'exécution.
La macro END est aussi utilisée pour terminer le programme.

PARAMETRES :

I Variable de contrôle pour la boucle
INIT Valeur initiale attribuée à I
STEP Valeur de l'incrément
LIMIT Valeur limite

14 - IFA

Macro utilisée pour comparer deux variables arithmétiques.

PARAMETRES :

VAR1 Première opérande
COND Opérateur (COND = EQ, NE, GT, LT, GE, LE)
VAR2 Deuxième opérande
LAB0 Adresse de branchement si la condition est vérifiée
LAB1 Adresse de branchement si la condition n'est pas vérifiée

15 - IFC et IFB

Macros utilisées pour comparer deux chaînes de caractères (bits)

PARAMETRES :

STR1 Premier opérande
I1 Position du premier caractère (bit) de STR1 à intervenir dans
 la comparaison
OPER Opérateur (OPER = EQ, NE)
STR2 Deuxième opérande
I2 Position du premier caractère (bit) de STR2 à intervenir dans la
 comparaison
LENGTH Nombre de caractères (bits) à comparer
LAB0 Adresse de branchement si la condition est vérifiée
LAB1 Adresse de branchement si la condition n'est pas vérifiée

16 - TEST B

Macro utilisée pour tester un ou plusieurs bits d'une chaîne.

PARAMETRES

NAME Nom de la chaîne
 I Position du premier bit à tester
 LENGIN Nombre de bits à tester
 LABO et LAB1 représentent des adresses de branchement de la même manière
 que pour les macros IFC et IFB

17 - READ et PRINT

Ces macros permettent de lire sur carte, ou d'imprimer les valeurs des différentes variables, selon un certain format.

PARAMETRES :

K1, K2, K3, K4 et K5 déterminent la nature de la variable à lire ou à écrire (D = décimale, C = caractère, B = bit)
 N1, N2, N3, N4, N5 Nombre de caractères à lire ou à imprimer
 A1, A2, A3, A4, A5 Adresses des variables à lire ou à imprimer

18 - READA et PRINTA

READA met dans l'accumulateur un certain nombre de caractères qui se trouvent sur une carte

PRINTA imprime le contenu de l'accumulateur selon un certain format

PARAMETRES :

K Format pour la lecture ou l'impression (D = décimal, H = hexadécimal, B = binaire, C = caractère)
 N Nombre de caractères à lire ou à imprimer
 Z Paramètre utilisé pour effectuer un passage à la carte ou à la ligne suivante (Z = S)
 NX Nombre de blancs à sauter pour la prochaine lecture ou impression.

19 - WEND1, WEND2, WEND3, WEND4, WEND5, WEND6 et WEND7

Ces macros ne sont jamais utilisées directement par le programmeur. Lors du dernier appel de la macro END, ces macros seront appelées automatiquement, pour générer les sous-programmes nécessaires au traitement des bits, caractères, tables, opérations de transfert etc....

DESCRIPTION INTERNE DU SYSTEME

TABLE DES MATIERES

CHAPITRE I Organisation Générale

1. Introduction.....	1
2. Organigramme Général.....	3

CHAPITRE II Description des principaux modules

1. MACGEN.....	7
2. MACLIB.....	25
3. CALLLIB.....	38
4. PRINTIT et PRINTER.....	68
5. CREATE.....	68
6. FETCH.....	71
7. CHANGE1 et CHANGE.....	72
8. DELETE1 et DELETE.....	73
9. Composition finale du système.....	73

CHAPITRE III

Conclusions.....	75
------------------	----

CHAPITRE I

ORGANISATION GENERALE

1 - INTRODUCTION

Le macro-générateur pour le TAS-86 a été presque entièrement écrit en PL/1, ce qui procure une plus grande facilité pour suivre sa logique.

Cependant, quelques unes de ses parties ont été écrites en assembleur, surtout pour permettre de manipuler les fichiers qui constituent la bibliothèque de macros système sur disque. En PL/1, il est impossible d'accéder directement à des fichiers partitionnés, ce qui est la façon la plus efficace d'organiser une bibliothèque de macros.

Le macro-générateur consiste en 11 procédures externes indépendantes dont les caractéristiques seront décrites en détail dans les paragraphes suivants.

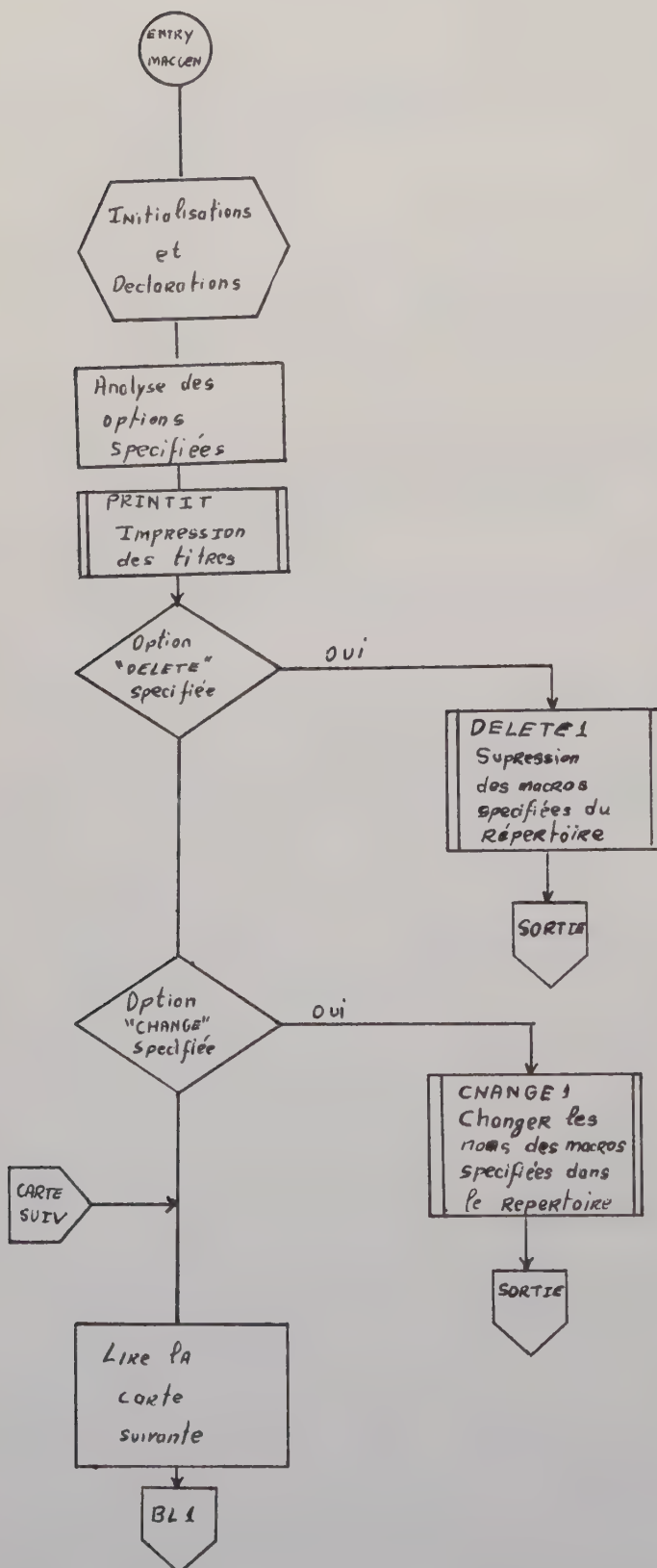
La liste des noms de procédures, accompagnées de leurs fonctions générales et tailles, figure ci-dessous :

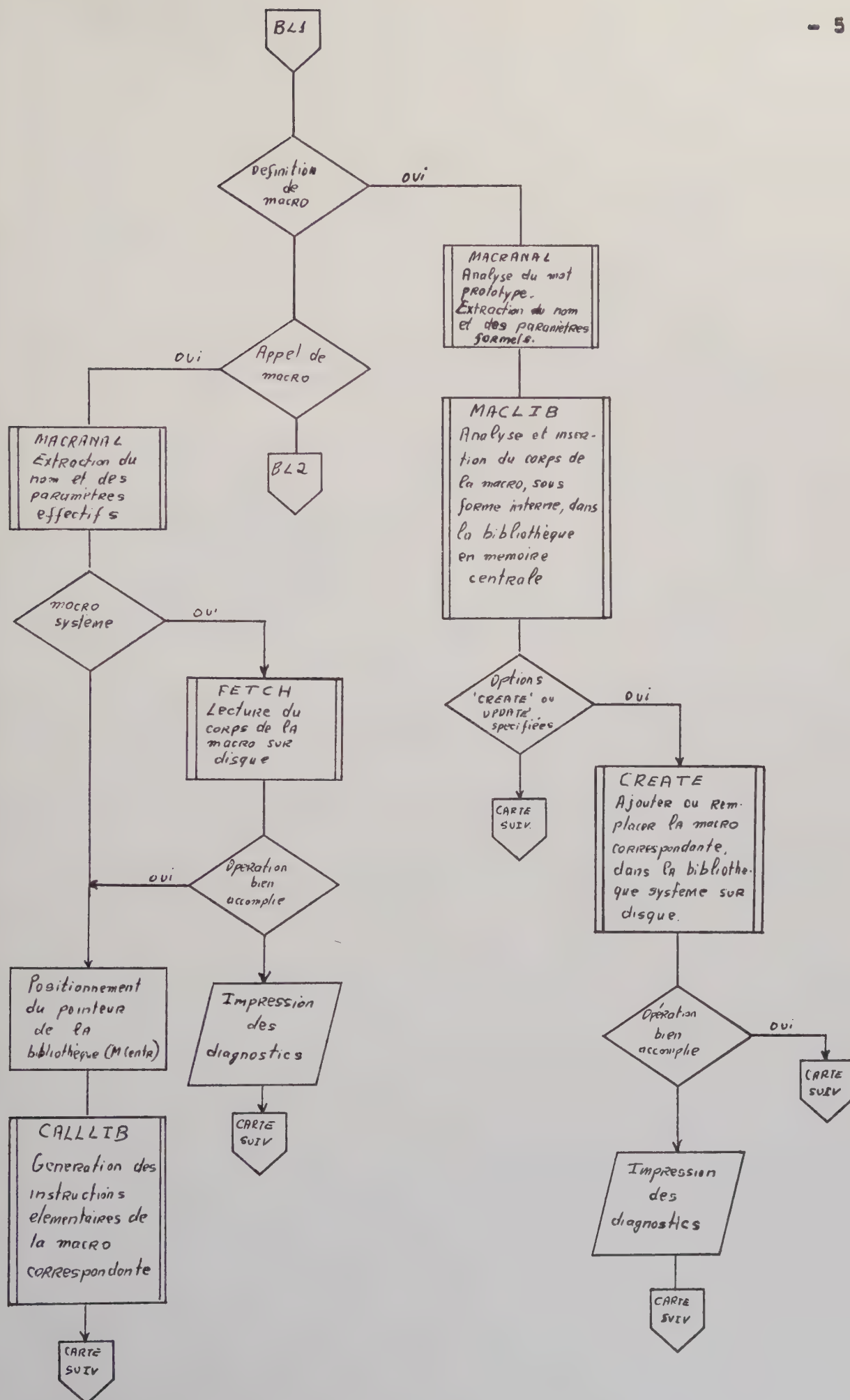
- a) MACGEN - PL/1 Nombre d'octets objets - 15062
- C'est la procédure qui prend l'option MAIN dans le système, étant donc son point d'entrée. Elle analyse le programme source et détermine les options à **utiliser par le** macro-générateur. A chaque fois qu'une définition ou un appel de macro sont détectés, **MACGEN** active les modules correspondants à poursuivre les analyses.

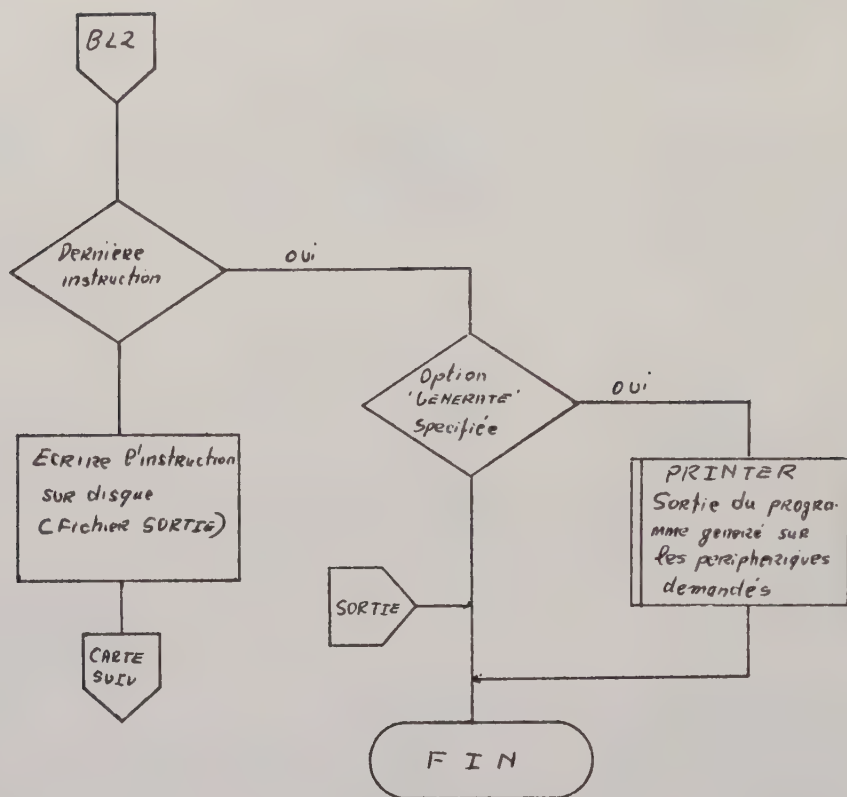
- b) MACLIB - PL/1 Nombre d'octets objets - 33816
C'est le module qui analyse les définitions de macros.
Le corps de la macro est laissé en mémoire centrale sous
une forme interne. Il est activé par MACGEN.
- c) CALLIB - PL/1 Nombre d'octets objets - 27002
Il est activé par MACGEN à chaque fois qu'un appel de macro
est détecté.
CALLIB est responsable du développement d'une macro ins-
truction dans ses instructions élémentaires.
Il suppose que le corps de la macro correspondante est
en mémoire centrale, lorsqu'elle est appelée.
- d) PRINTIT - PL/1 Nombre d'octets objets - 966
Il imprime simplement les titres et les options correspon-
dantes spécifiées au macro-générateur.
Il est activé par MACGEN.
- e) PRINTER - PL/1 Nombre d'octets objets - 2150
Ce module est activé par MACGEN lorsque la génération est
terminée. Il analyse les options spécifiées et sort le
programme traduit sur les périphériques déterminés.
- f) CHANGE1 - PL/1 Nombre d'octets objets : 1578
Il est activé par MACGEN lorsque l'option "CHANGE" est
spécifiée. Il analyse le champ PARM de la carte EXEC et
extraît les anciens et les nouveaux noms des macros qui
devront être changés.
- g) DELETE1 - PL/1 Nombre d'octets objets : 1062
Il est activé par MACGEN lorsque l'option "DELETE" est
spécifiée. De la même façon que "CHANGE", il analyse le
champ PARM et extrait les noms de macros à supprimer dans
la bibliothèque.

- h) CHANGE - Assembleur Nombre d'octets objets : 272
Il est activé par CHANGE1 ayant comme paramètres les anciens et les nouveaux noms de macros.
En utilisant la macro STOW (OS 360 MACRO), il altère les noms dans le répertoire de la bibliothèque.
- i) FETCH - Assembleur Nombre d'octets objets : 440
FETCH accède à la bibliothèque qui se trouve sur disque et place le corps de la macro correspondante à l'endroit correct, en mémoire centrale.
Il est activé soit par MACGEN soit par CALLLIB (dans le cas d'appels emboîtés) à chaque fois qu'un appel de macro système est détecté.
- j) CREATE - Assembleur Nombre d'octets objets : 552
C'est le module qui ajoute ou remplace une définition de macro sur disque.
Il est activé par MACGEN si les options "CREATE" ou "UPDATE" sont spécifiées.
- k) DELETE - Assembleur Nombre d'octets objets : 248
Il est activé par DELETE1 ayant comme paramètre un nom de macro qui devra être effacé du répertoire.

2 - ORGANIGRAMME GENERAL -







CHAPITRE II

DESCRIPTION DES PRINCIPAUX MODULES

1 - MACGEN

1-1 - Description Générale

Comme nous l'avons signalé dans le Chapitre I, MACGEN est le point d'entrée du macro-générateur.

Il analyse le champ PARM de l'instruction EXEC qui active le macro-générateur et qui a la forme :

```
// EXEC MACGEN, PARM = 'GENERATE, DECK...'
```

MACGEN est également le nom de la procédure cataloguée qui invoque le macro-générateur.

Toutes les cartes de définition de fichier requises, sont définies dans la procédure cataloguée.

Après l'analyse du champ PARM, PRINTIT est activé pour imprimer les titres et la liste des options. Les cartes qui constituent le programme source sont alors lues, une par une, dans une zone d'analyse (ZONE). Les noms entre parenthèses indiquent les dénominations par lesquelles les variables mentionnées sont déclarées dans le programme PL/1. Dans ce cas, par exemple, la

zone d'analyse est appelée ZONE. Il s'agit d'un tampon qui peut contenir jusqu'à 864 caractères. Une telle longueur est nécessaire lorsque des cartes de séquence sont ajoutées à la première.

Une description détaillée de l'usage de chaque variable se trouve, sous la forme de commentaires, dans les listings du programme. Dans ce texte nous ne nous référons qu'aux plus importantes. ZONE est analysée, caractère par caractère, de façon à déterminer l'action à prendre.

Un organigramme général de MACGEN est donné dans la section 2 du Chapitre I.

Si une définition ou un appel de macro sont détectés, avant de passer le contrôle à MACLIB ou à CALLLIB, MACGEN active une procédure interne MACRANAL qui analyse soit le mot prototype (dans le cas d'une définition) soit l'instruction d'appel. MACRANAL extrait le nom de la macro et ses paramètres. Sa description apparaîtra plus tard, dans un autre item. Si, par hasard, ni une définition, ni un appel sont détectés, MACGEN pose le contenu de la carte dans le fichier de sortie. Lorsque la fin du programme est détectée, MACGEN active PRINTER de façon à faire sortir le programme traduit sur les périphériques demandés.

En revenant de PRINTER, le programme atteint la sortie. Au cours de la génération, le macro-générateur place le programme généré dans un fichier de travail, organisé séquentiellement, sur disque (SORTIE), ou la longueur de chaque bloc est de 80 octets.

Après le retour des programmes écrits en assembleur et qui accèdent à la bibliothèque de macros sur disque (la bibliothèque de macros système est cataloguée sous le nom de SYS1.TR86MGEN), MACGEN interprète le résultat des opérations et imprime les diagnostics correspondants.

1-2 - Options prévues pour l'utilisateur

L'utilisateur peut spécifier un certain nombre d'options au macro-générateur. Leurs différents sens seront expliqués ci-dessous.

Le contenu du champ PARM dans l'instruction EXEC qui invoque le macro-générateur, est automatiquement placé dans une zone de 50 caractères (nommée PARM, dans le programme). Cette zone est analysée afin de positionner quelques indicateurs qui préciseront les options en usage.

L'état normal des indicateurs d'options est zéro. Cette valeur est associée à l'option prise par défaut pour chaque cas.

L'utilisateur doit se rappeler qu'un maximum de 50 caractères est admis dans le champ PARM.

Les différentes options sont :

a) SOURCE -NOSOURCE

Si l'option NOSOURCE est spécifiée, le programme d'entrée ne s'imprime pas. L'option prise par défaut est SOURCE.

L'indicateur associé est SOURIND.

b) GENERATE - CREATE - UPDATE - CHANGE - DELETE

Ces options commandent les principales opérations du macro-générateur.

GENERATE est utilisée dans les cas habituels. le système analyse des éventuelles définitions de macros laissées en mémoire centrale et génère les appels dans ses opérations élémentaires.

Les macros système sont lues du disque en mémoire centrale, lorsqu'il est nécessaire.

CREATE est utilisée pour introduire des nouvelles macros dans la bibliothèque sur disque. La définition est analysée et transformée dans une forme interne par MACLIB, et, ensuite ajoutée à la bibliothèque.

Un nombre quelconque de macros peut être inséré en un seul passage. La seule limitation est la place disponible sur disque.

UPDATE est utilisée pour altérer une définition déjà existante dans la bibliothèque qui est automatiquement remplacée par une nouvelle version. Comme pour le cas de CREATE, un nombre quelconque de macros peut être remplacé en un seul passage.

CHANGE altère des noms dans le répertoire. Le moyen de spécifier les anciens et les nouveaux noms sera expliqué plus loin.

DELETE, supprime simplement des noms de macros du répertoire. Toutes ces possibilités sont mutuellement exclusives et l'option prise par défaut est GENERATE.

Les différentes options spécifiées doivent être séparées par des virgules, dans le champ PARM de la carte job EXEC.

Exemple : / / EXEC MACGEN, PARM = 'SOURCE, UPDATE'

Dans les cas spéciaux de DELETE et CHANGE, le champ PARM doit avoir la forme suivante :

PARM = 'DELETE : MAC1, MAC2, ... MACN'

Les noms de macros à supprimer doivent être séparés par des virgules. Des blancs ne sont pas admis.

PARM = 'CHANGE : MAC1 → MAC2, MAC3 → MAC4'

La macro nommée MAC1 deviendra MAC2 et MAC3 sera nommée MAC4. Dans les deux cas, le champ PARM ne peut pas contenir plus de 50 caractères.

L'indicateur associé à toutes ces options est CREIND.

c) LIST - NOLIST

Si NOLIST est spécifié, le programme généré ne sort pas sur imprimante. L'option prise par défaut est LIST.

L'indicateur associé est LISTIND.

d) DECK - NODECK

Si NODECK est spécifié, le programme généré n'est pas perforé sur cartes. Si rien n'est indiqué, DECK est pris par défaut. L'indicateur associé est NOPUNCH.

e) IDENT - NO-IDENT

Si le programmeur spécifie l'option IDENT, les trois premiers caractères du nom du programme (l'opérande de l'instruction NM) sont perforés dans les colonnes 73, 74 et 75 de toutes les cartes du programme généré. Le mécanisme est utilisé dans le but d'identifier le paquet correspondant. L'option prise par défaut est NO-IDENT.

L'indicateur associé est IDEIND.

Un exemple typique du contenu du champ PARM est :

```
// EXEC MACGEN, PARM = 'GENERATE, IDENT, DECK'
```

Si aucune option n'est spécifiée, celles supposées sont :

```
GENERATE  
SOURCE  
LIST  
DECK  
NO-IDENT
```

1-3 - Description de la procédure MACRANAL

MACRANAL est une procédure interne pour MACGEN. Elle est activée à chaque fois qu'une définition ou un appel de macro sont détectés. Elle analyse soit le mot prototype, soit l'instruction d'appel afin d'en extraire des informations de base.

Ses principales fonctions sont décrites dans les items suivants.

1-31 - Définition de macro

Dans le cas d'une définition de macro, MACRANAL analyse les mots prototypes comme :

```
$LAB = $ADD $PAR1,$PAR2,$PAR3
```

Tous les paramètres formels sont extraits et introduits dans la table d'arguments (ARGUM). La table d'arguments est déclarée comme ARGUM (0 : 15, 10) CHARACTER (7)

La valeur 15 de la première dimension résulte du nombre maximum de paramètres permis dans une définition de macro.

La deuxième dimension résulte du niveau maximum d'emboîtement qui est de 10.

MACRANAL remplit la table d'arguments en conservant la deuxième dimension égale à 1.

Les autres parties sont remplies par CALLLIB (procédure responsable de la génération d'instructions élémentaires), lorsque des appels emboîtés sont détectés.

Dans le cas d'un mot prototype comme :

\$LAB = ADD \$PAR1, \$PAR2, \$PAR3

le résultat final est :

ARGUM		1	2	3	10
0 1 2 3 15 (PTARG)	1	LAB			
	1	PAR1			
	2	PAR2			
	3	PAR3			
	15				

fig. 1

Le pointeur pour la première dimension est PTARG.

Le contenu de ARGUM est examiné à chaque fois qu'un paramètre formel est détecté, au cours du traitement de la

définition de la macro par MACLIB.

Le nom de la macro est extrait par MACRANAL et placé dans le répertoire (DIRECTO), qui se trouve en mémoire centrale.

Le répertoire contient des entrées pour les noms de macros et leurs relatives localisations en mémoire centrale.

Son organisation est :

DIRECTO



nom de la macro	pointeur de la bibliothèque	SWITCH
BEGIN	1	0
ADD	10	0

fig. 2

A chaque fois qu'une macro est définie, son nom s'introduit comme entrée dans le répertoire. En même temps, la valeur actuelle du pointeur de la bibliothèque est associée à ce nom.

De cette manière, lorsque le nom d'une certaine macro sera trouvé dans le répertoire, la valeur correspondante du pointeur indiquera l'entrée de la bibliothèque qui contient la première instruction de la macro correspondante.

Si, par exemple, la macro ADD est placée dans la bibliothèque à partir de l'entrée 10, le répertoire sera représenté par la figure 2.

Nous supposons qu'une certaine macro nommée BEGIN a été définie avant ADD.

Le pointeur associé au répertoire (DIRECTO) est PTDIR.

L'utilisation du paramètre SWITCH sera expliquée plus tard.

Le répertoire construit par MACRANAL s'applique aux macros définies par le programmeur et reste tout le temps en mémoire centrale. Son contenu est perdu à la fin du passage.

Le répertoire pour les macros système existe sur disque.

Lorsque les options CREATE ou UPDATE sont indiquées, aucune entrée n'est ajoutée au répertoire en mémoire centrale.

Le nombre maximum d'entrées prévues dans DIRECTO est de 20.

1-32 - Appel de macro

Ainsi que pour la définition de macros, MACRANAL est activée à chaque fois qu'un appel de macro est détecté. Dans le cas d'un appel, un indicateur (MCCALL) est positionné à 1.

MACRANAL extrait le nom de la macro et ses paramètres effectifs. Ces paramètres sont placés dans la table d'arguments (ARGUM), de la même manière que dans le cas précédent.

Lorsqu'elle génère les instructions élémentaires, la procédure CALLLIB accède à la table d'arguments.

MACRANAL examine le répertoire en mémoire centrale, pour le nom de la macro. S'il contient l'entrée correspondante, la valeur du pointeur de la bibliothèque est extraite et passée à CALLLIB.

Comme nous l'avons déjà signalé, la valeur du pointeur indique l'entrée dans la bibliothèque où est située la première instruction de la macro correspondante.

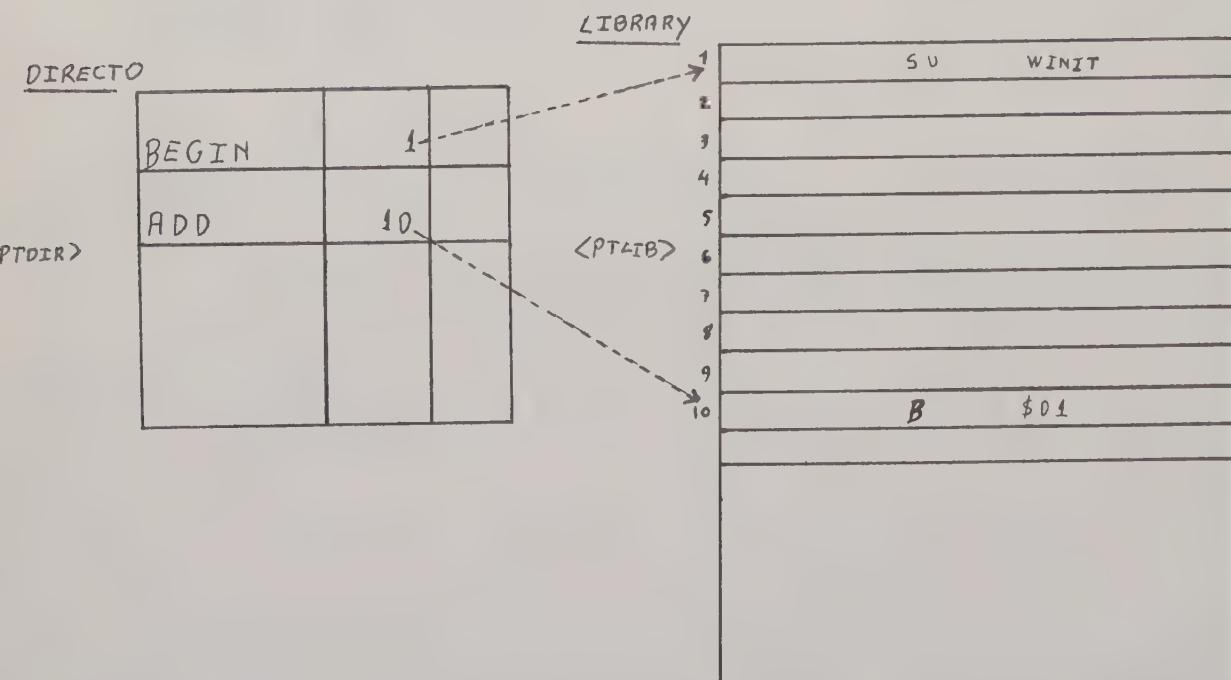


fig. 3

LIBRARY est une table de 500 entrées. Chaque entrée est longue de 80 caractères.

Si le nom d'une macro ne se trouve pas dans DIRECTO, la procédure FETCH est activée pour examiner le répertoire sur disque. S'il existe, FETCH place le corps de la macro à partir de la première localisation libre dans LIBRARY.

Un indicateur (EXTRN) est positionné à 1, afin de signaler une macro système.

Le macro-générateur gère d'une façon dynamique l'espace disponible dans LIBRARY. A chaque fois que la génération d'une macro système est terminée, l'espace qu'elle occupait dans LIBRARY est libéré.

Si des appels emboîtés sont détectés, des macros internes se placent automatiquement dans la prochaine position libre. Ce mécanisme est géré par une pile du genre "last in first out".

Cette opération est décrite dans les paragraphes suivants. La pile s'appelle DYNCORE (mémoire centrale dynamique) et son pointeur est PTODYN.

Le macro-générateur exige que toutes les éventuelles définitions de macros soient placées avant la première instruction d'appel de macro.

Lorsque le premier appel est détecté, la valeur de PTLIB (la prochaine localisation libre dans LIBRARY) se place dans la pile. La valeur de PTODYN est alors augmentée de 1 pour provoquer la descente de pile.

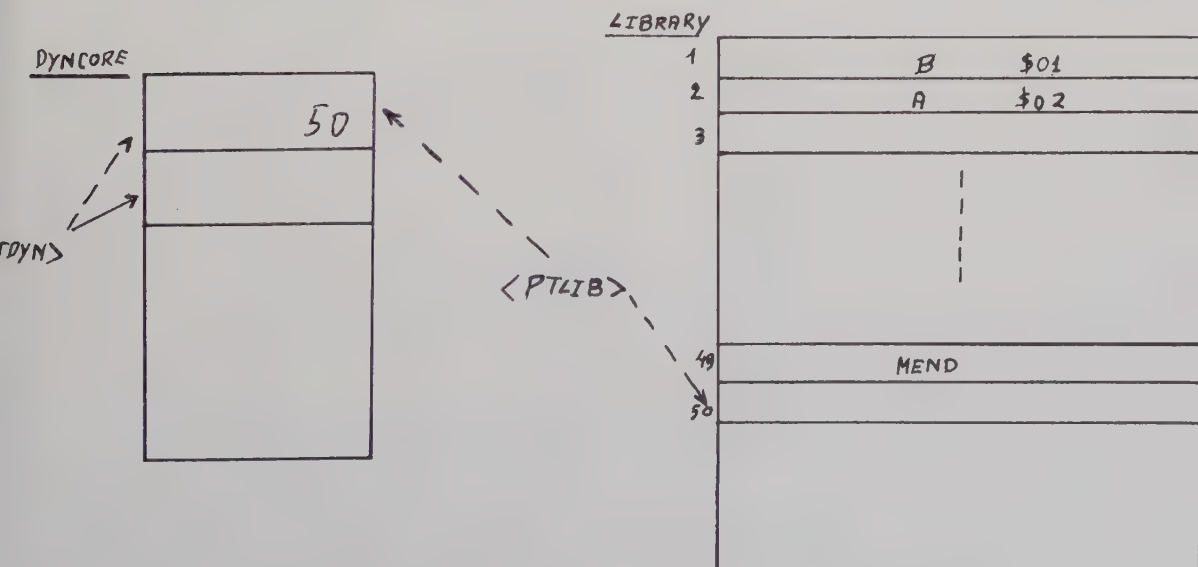


fig. 4

La prochaine localisation libre dans LIBRARY est empilée en DYNCORE.

La valeur de DYNCORE (PTDYN - 2), dans le cas 50 est passée à CALLLIB comme l'adresse de la première instruction de la macro READ.

Si READ ne contient aucun appel emboîté, lorsqu'elle termine, une montée de pile est générée et les localisations qu'elle occupait dans LIBRARY deviennent libres.

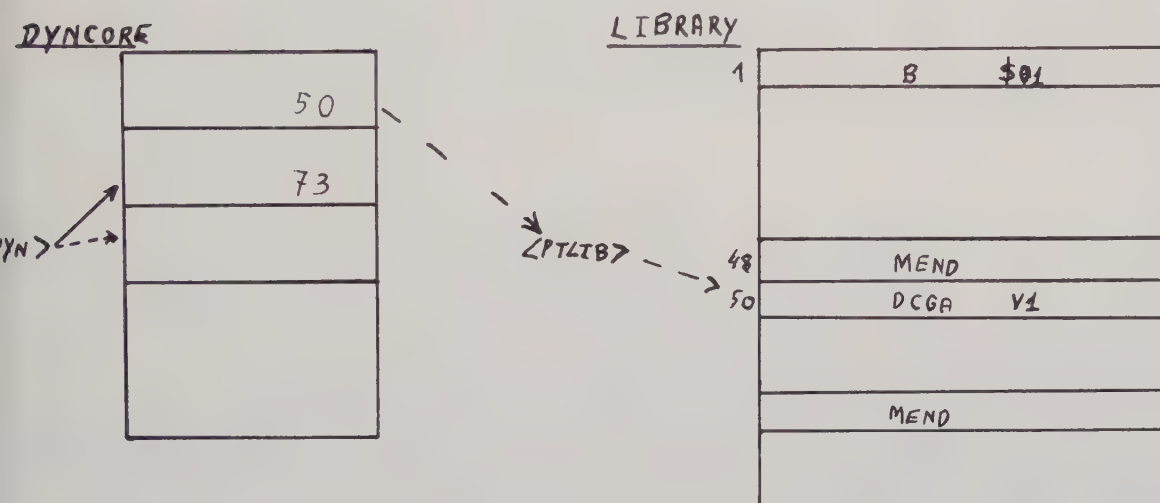


fig. 6

Si une autre macro système est appelée, elle se placera à partir de l'entrée 50 et le même mécanisme se répétera.

Cependant, si la macro READ génère un appel emboîté, une descente de pile se produit de nouveau et la prochaine macro se place immédiatement après la dernière instruction de READ (localisation 73)

DYNCORE

LIBRARY

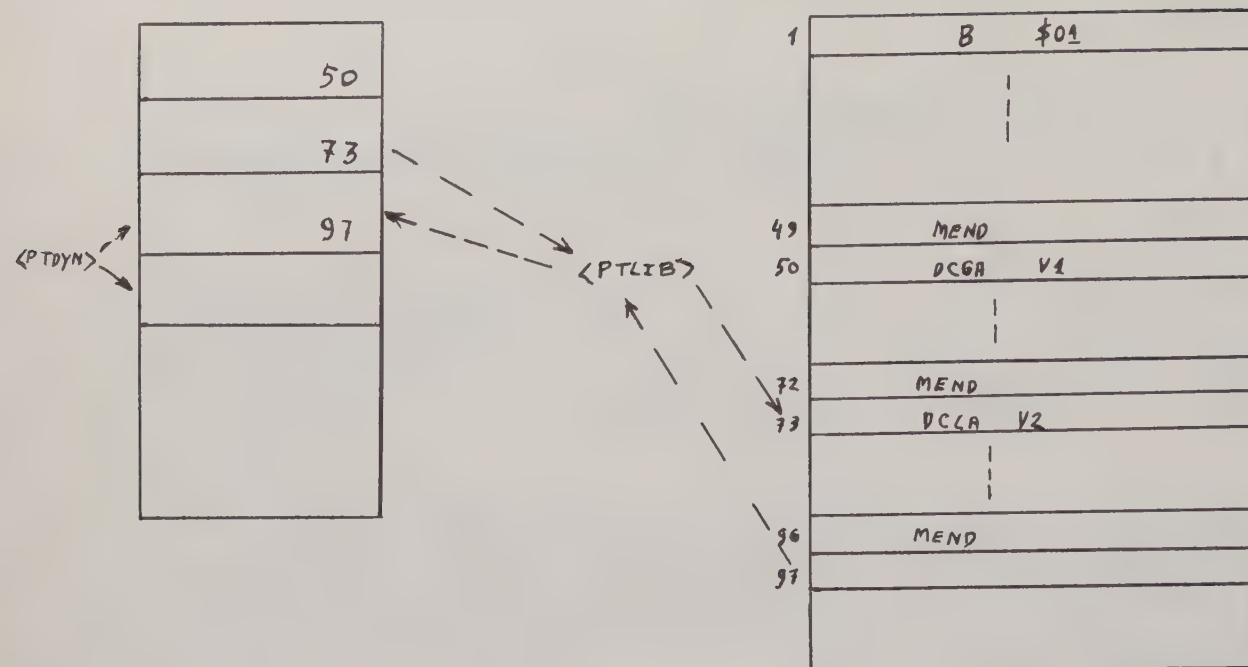


fig. 7

Chaque fois qu'un appel emboîté est détecté, le processus se poursuit automatiquement.

La méthode, décrite ci-dessus, pour gérer d'une façon dynamique les allocations d'espace en mémoire, quoique simple, est très puissante et peut être utilisée dans le contrôle des allocations mémoire pour des structures comme les blocs BEGIN-END employées en PL/1 et ALGOL.

Si les options 'CREATE' ou 'UPDATE' sont spécifiées, immédiatement après avoir écrit le corps de la macro correspondante sur disque, PTLIB est repositionné à 1. Telle est la raison du nombre illimité de macros qui peuvent être introduites ou remplacées sur disque dans un seul passage.

Le nombre d'instructions par macro est limité à 500 (nombre d'entrées dans LIBRARY) qui est plus que suffisant.

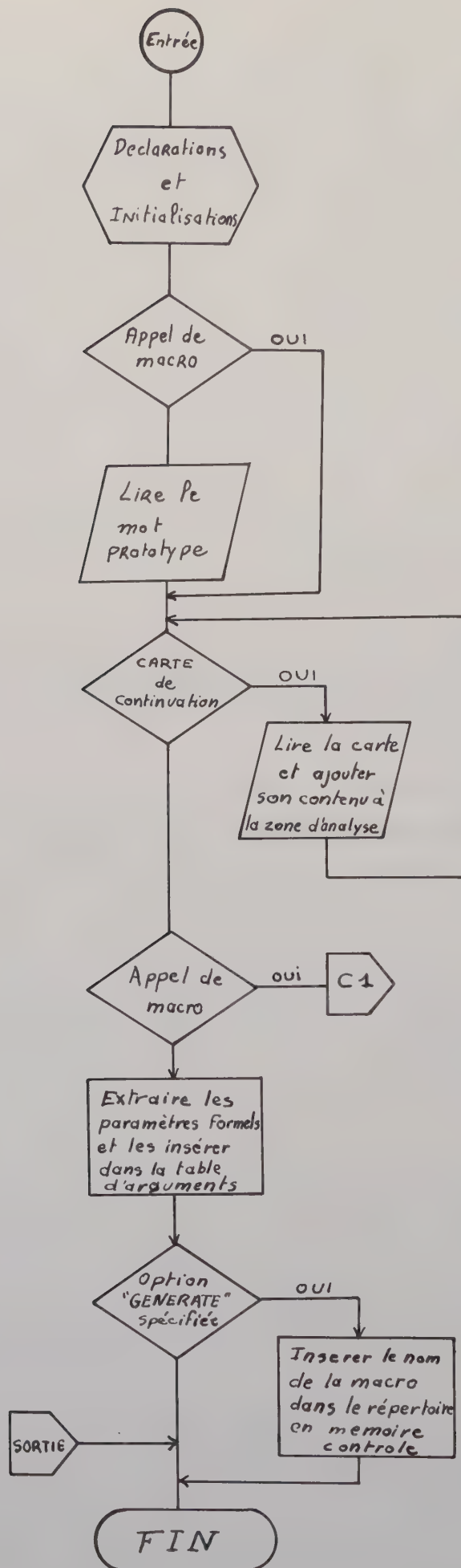
1-32 - Organigramme général

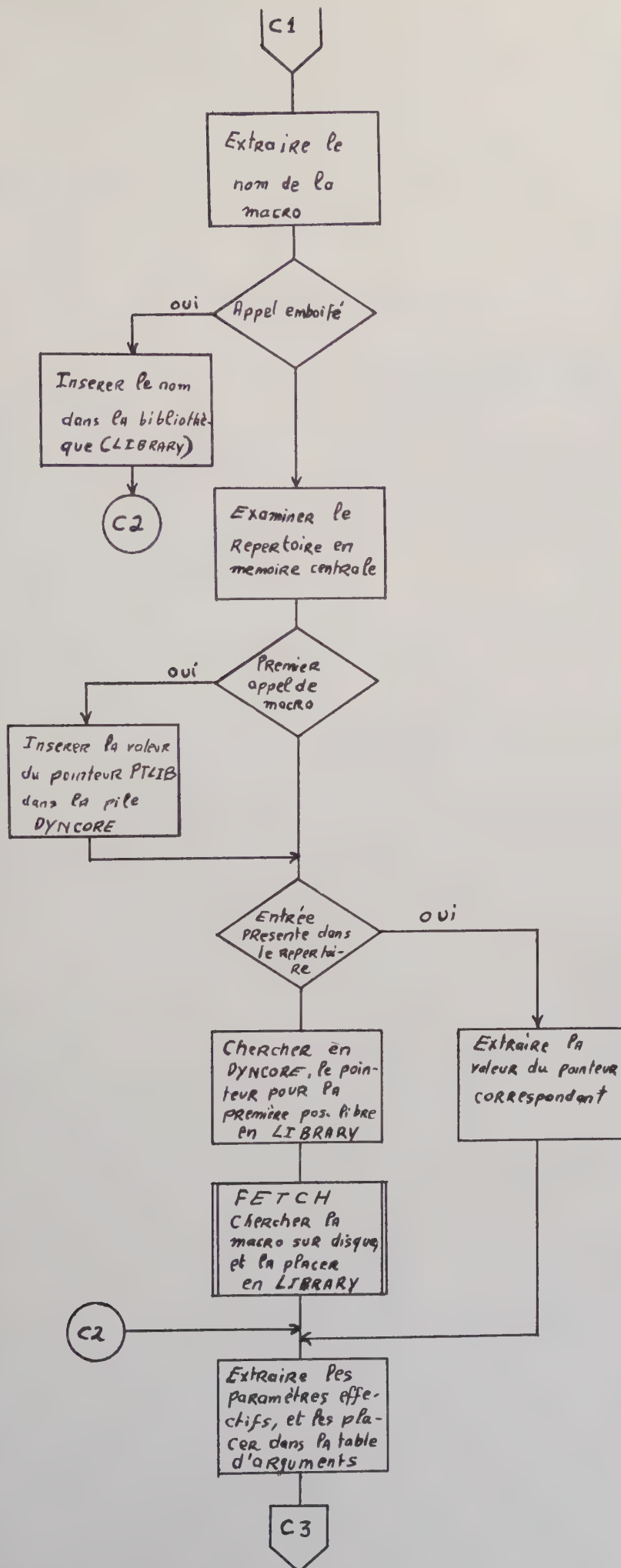
La méthode d'extraction de paramètres varie légèrement dans le cas d'appels de macros étant donné que MACRANAL n'examine pas les caractères qui constituent les arguments effectifs. Elle ne fait qu'extraire 7 caractères de chaque argument en les plaçant ensuite dans la position correspondante de ARGUM. Cela a été conçu afin de permettre une plus grande souplesse dans la nature des arguments effectifs.

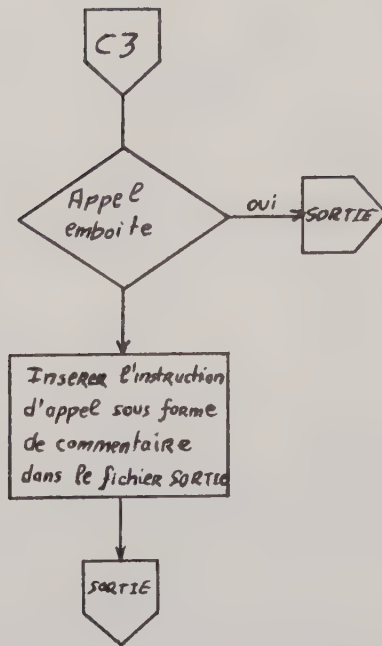
Lorsqu'un appel emboîté est détecté dans une définition de macro, MACRANAL est activé par MACLIB.

Dans ce cas, un indicateur (NEST) est positionné à 1. Les arguments sont placés en ARGUM de la même manière, mais, le nom de la macro est simplement introduit dans l'entrée correspondante de la bibliothèque.

Durant la génération de la macro, CALLLIB sera chargé soit d'extraire le nom pour examiner le répertoire, soit d'activer FETCH pour le chercher sur disque.







2 - MACLIB

2-1 - Description Générale

MACLIB est la procédure qui analyse les définitions de macros. Elle effectue une analyse syntaxique générale des instructions source qui constituent le corps de la macro et les insère dans la bibliothèque qui est en mémoire centrale (LIBRARY), sous une forme interne de manière à permettre l'accès par CALLLIB, à chaque fois qu'elle est appelée.

Les principales fonctions accomplies par MACLIB sont décrites dans les paragraphes suivants.

2-2 - Paramètres formels

A chaque fois que MACLIB détecte un paramètre formel, dans ZONE (zone d'analyse) il en est extrait et placé dans RECORD (variable longue de 7 caractères).

Une procédure interne SEARARG (recherche de l'argument) est appelée pour examiner la table d'arguments, afin de détecter la présence du paramètre formel correspondant. S'il existe, SEARARG place dans RECORD le caractère "\$", suivi de deux chiffres qui correspondent à l'ordre de l'argument.

Ex :

```
MACRO
ADD   $PAR1, $PAR2, $PAR3
B     $PAR1
A     $PAR2
etc.
```

Lorsque l'analyse de l'opérande de l'instruction A \$PAR2 s'effectue, PAR2 se place dans RECORD. SEARARG est alors appelée et retourne \$02.

Ainsi, l'instruction ADD, sous sa forme interne, est représentée par :

A \$02

2-3 Variables internes

MACLIB détecte une variable interne par deux "\$" précédant l'identificateur. A ce stade, le nom de la variable interne reste immuable, précédé d'un seul "\$".

Toutes les actions qui se rapportent aux variables internes comme déclarations, instructions set, etc. sont effectuées par CALLLIB.

La tâche de MACLIB est de grouper les instructions correctement dans la bibliothèque.

Une instruction du type :

\$\$VAR = SETA \$PAR1+\$\$VAR

est analysée par MACLIB et placée dans la bibliothèque avec le format (nous supposons que PAR1 est le premier paramètre de la liste, dans le mot prototype) :

\$VAR SETA \$01+\$VAR

2-4 - Etiquettes Internes

Les étiquettes internes sont détectées et placées dans une table de symboles nommée LABELTAB. A chaque nom en LABELTAB nous faisons correspondre une adresse qui précise l'entrée dans la bibliothèque (LIBRARY) où l'étiquette interne est définie.

Ex :

```
MACRO
TEST  $PAR1,$PAR2
DCLC  VAR

CAB ($PAR1 EQ ' ').LAB

.LAB = NOP
```

Si la valeur de PTLIB correspondante à l'instruction NOP est 18, l'entrée générée en LABELTAB sera :

LABELTAB

LABNAME	LABADDR
LAB	18

<PTLAB> →

fig. 8

LABELTAB fonctionne comme une table symbolique ordinaire ou PTLIB représente le compteur d'adresses.

Lorsque la définition de la macro atteint la fin, ce sera le moment d'accéder à LABELTAB afin de résoudre les branchements internes spécifiés dans les instructions CAB et UAB.

Le nombre d'entrées en LABELTAB est limité à 100, ce qui limite également le nombre d'étiquettes internes qui peuvent être définies dans une macro.

2-5 - Instructions CAB et UAB

Dans l'instruction CAB, MACLIB teste la syntaxe de la partie opérande, la validité de l'opérateur de comparaison et remplace les paramètres formels et les variables internes par leur représentation interne.

Lorsque l'on analyse une instruction CAB ou UAB, il n'est pas toujours possible de résoudre le branchement interne spécifié par l'étiquette interne car elle peut apparaître définie quelque part plus loin, dans le programme.

Il s'agit d'un problème classique, résolu par un second passage dans la majorité des assembleurs.

En ce qui concerne le macro-générateur, un second passage ne se justifie pas si ce n'est que pour résoudre des branchements internes.

Pour résoudre ce problème, une méthode rapide a été conçue.

Après l'analyse de l'instruction CAB ou UAB, la valeur courante de PTLIB est insérée dans une table (BRANCHCONT) qui mémorise les adresses des instructions CAB et UAB.

En même temps que la valeur de PTLIB, un pointeur pour la position du premier caractère de l'étiquette interne dans l'instruction CAB ou UAB est introduit dans BRANCHCONT.

```
Ex :                               position 31
      CAB ($PAR1 EQ ' '). LAB
```

valeur courante de PTLIB - 23

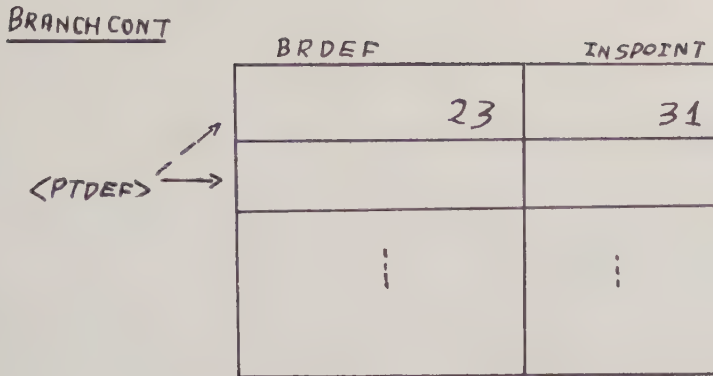


fig. 9

A la fin de la définition, le contenu de BRANCHCONT est examiné. Les valeurs de PTLIB correspondant à des instructions de branchement sont alors utilisées, conjointement avec le pointeur de position, pour extraire le nom de l'étiquette interne.

Il faut ensuite, chercher en LABELTAB l'entrée où l'étiquette interne correspondante est définie.

Si elle existe, la différence entre son adresse et la valeur extraite de BRANCHCONT (adresse de l'instruction qui s'y réfère) est calculée.

Cette différence, convertie sous forme de caractère, est alors insérée à la fin de la partie opérande des instructions CAB ou UAB en question.

Si dans l'exemple précédent, LAB se référait à une étiquette définie 12 instructions plus loin, la représentation interne de l'instruction CAB serait :

CAB (\$01 EQ ' ').LAB 12

Au cours de la génération, si un branchement doit être exécuté, CALLLIB va extraire la valeur 12 et l'ajouter à la valeur courante de PTLIB.

2-6 - Instructions set

MACLIB analyse les instructions SETA et SETC et teste si les opérandes et opérateurs sont **cohérents**.

Comme nous l'avons déjà signalé, à ce stade, aucune action ne peut être entreprise pour altérer les valeurs des variables internes. MACLIB ne fait que tester la validité de l'instruction et l'insérer en bibliothèque.

Durant la génération, CALLLIB considère que toutes les instructions en bibliothèque sont syntaxiquement correctes.

Une instruction du type :

```
$$CHAR = SETC $PAR&'.'
```

après avoir été analysée, est insérée comme :

```
$CHAR SETC $01&'.'
```

Nous supposons que PAR est le premier paramètre formel de la liste.

2-7 - Appels emboîtés

Lorsque, dans une définition de macro, un appel emboîté est détecté, le nom de la macro et les arguments sont groupés de façon à ce que l'on puisse y accéder, durant la génération.

Les paramètres formels de la macro externe sont gardés dans la deuxième colonne de ARGUM et la valeur du pointeur (PTARG) dans PTARGT. Après ces mesures, MACRANAL est appelée pour analyser l'instruction d'appel. Cependant, à ce stade, les répertoires ne sont pas examinés pour détecter si la macro interne a déjà été définie ou non.

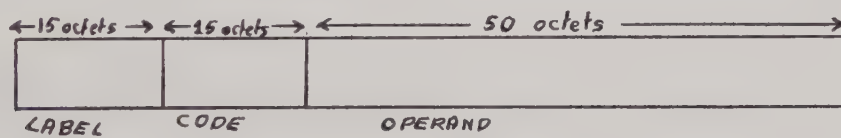
Cette procédure permet au programmeur d'inclure librement des appels emboîtés dans une définition de macro.

Au cours de la génération, CALLLIB va faire des investigations sur l'existence de la macro interne.

MACRANAL ne fait qu'insérer le nom de la macro interne, précédé d'un astérisque, dans la bibliothèque.

En revenant de MACRANAL, les conditions initiales sont rétablies et les paramètres extraits de la macro interne sont placés dans la table ARGUM1.

Comme les appels emboîtés présentent des paramètres formels et effectifs interposés, ils sont analysés un par un par MACLIB. Chaque entrée en bibliothèque (LIBRARY) peut être considérée comme étant divisée en 3 parties différentes :



LABEL et CODE sont longs de 15 caractères, chacun ; OPERAND contient 50, ce qui fait un total de 80.

Le nom de la macro interne, précédé d'un astérisque, se place à partir de la première position de CODE. Les paramètres se placent séquentiellement dans OPERAND, séparés par des virgules.

Lorsque des paramètres formels sont détectés, SEARARG est appelé pour indiquer leurs positions dans la liste.

Les paramètres effectifs restent immuables.

Lorsque tous les paramètres ont été introduits dans OPERAND, une marque de fin de liste y est insérée (',/').

Si 50 caractères ne sont pas suffisants pour représenter la liste entière, les restants seront placés dans la prochaine entrée de la bibliothèque, en commençant à la partie opérande (OPERAND).

Ex :

```
MACRO
$LAB = LEVEL1 $PAR1, $PAR2, $PAR3, $PAR4
      B $PAR1
      A $PAR3
$LAB = *LEVEL2 $PAR3, ZONE, , $PAR1, VAR

MEND
```

L'entrée de la bibliothèque qui contient l'appel emboîté a la forme suivante :

LABEL	CODE	OPERAND
\$00	*LEVEL2	\$03, ZONE, b, \$01, VAR, /

Si la liste de paramètres remplit exactement les 50 positions de OPERAND, une barre "/" sera placée dans la dernière position de CODE pour indiquer la fin de la liste.

Le paramètre manquant est remplacé par un blanc qui sera détecté par CALLLIB. Comme nous le voyons, des virgules sont utilisées dans la liste, pour séparer un paramètre de l'autre.

Etant donné que CALLLIB ne cherchera pas des paramètres entre apostrophes (comme le fait MACRANAL), les virgules ne peuvent pas faire partie des paramètres effectifs figurant dans un appel emboîté.

Si la macro interne ne contient aucun paramètre, la marque de fin de liste est insérée dans les positions 2 et 3 de OPERAND.

2-8 Déclarations de variables internes

La partie opérande des instructions de déclaration est analysée pour **vérifier** que tous les identificateurs des variables internes commencent par une lettre et que le nombre de caractères de chaque identificateur ne dépasse pas 6.

S'il n'y a pas d'erreurs détectées, la liste de variables internes à déclarer est insérée dans la bibliothèque.

Ex :

	DCLC	CHAR, CHAR1
--	------	-------------

2-9 Commentaires

Lorsque les commentaires se placent séparément dans une carte, ils sont ignorés et par conséquent, exclus de la bibliothèque.

D'un autre côté, si les commentaires suivent la partie opérande d'une instruction, ils sont alors insérés. Dans ce cas ils seront reproduits durant la génération

Ex :

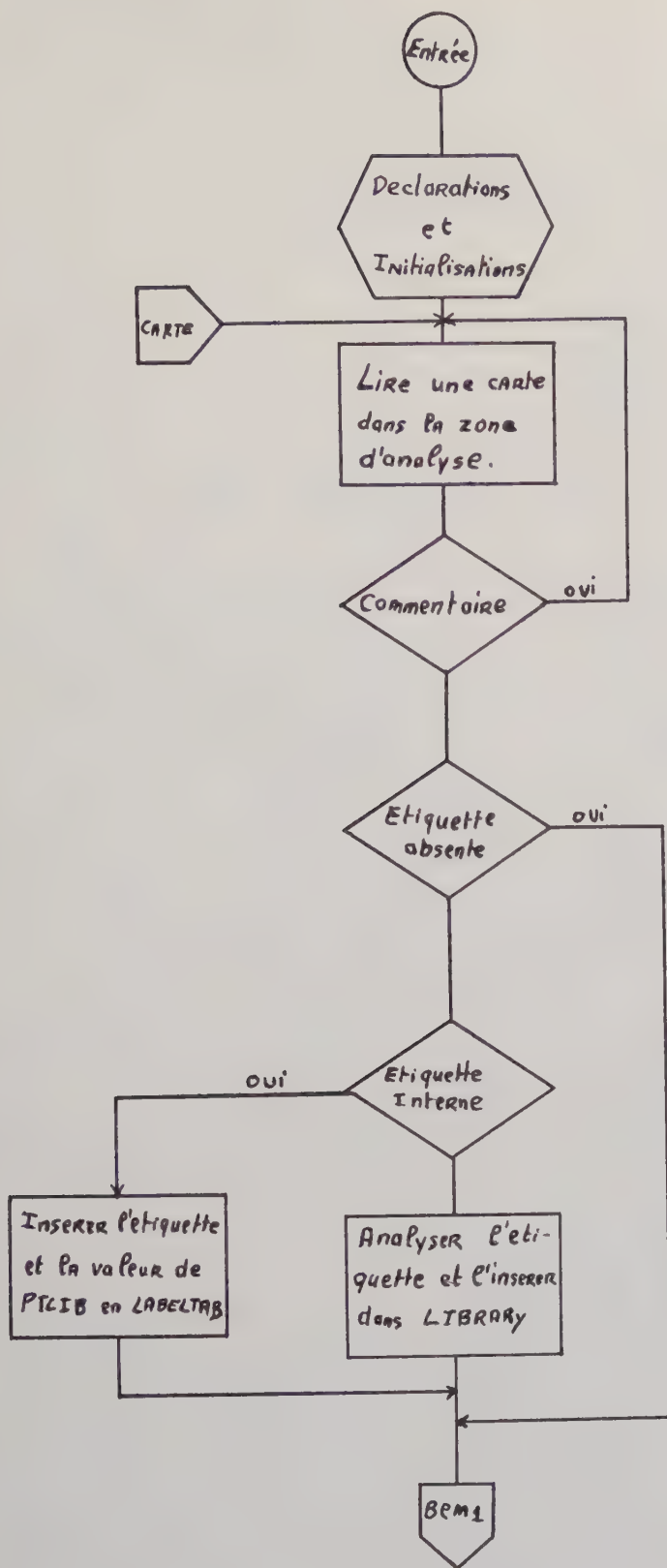
- - CECI EST UN COMMENTAIRE - -

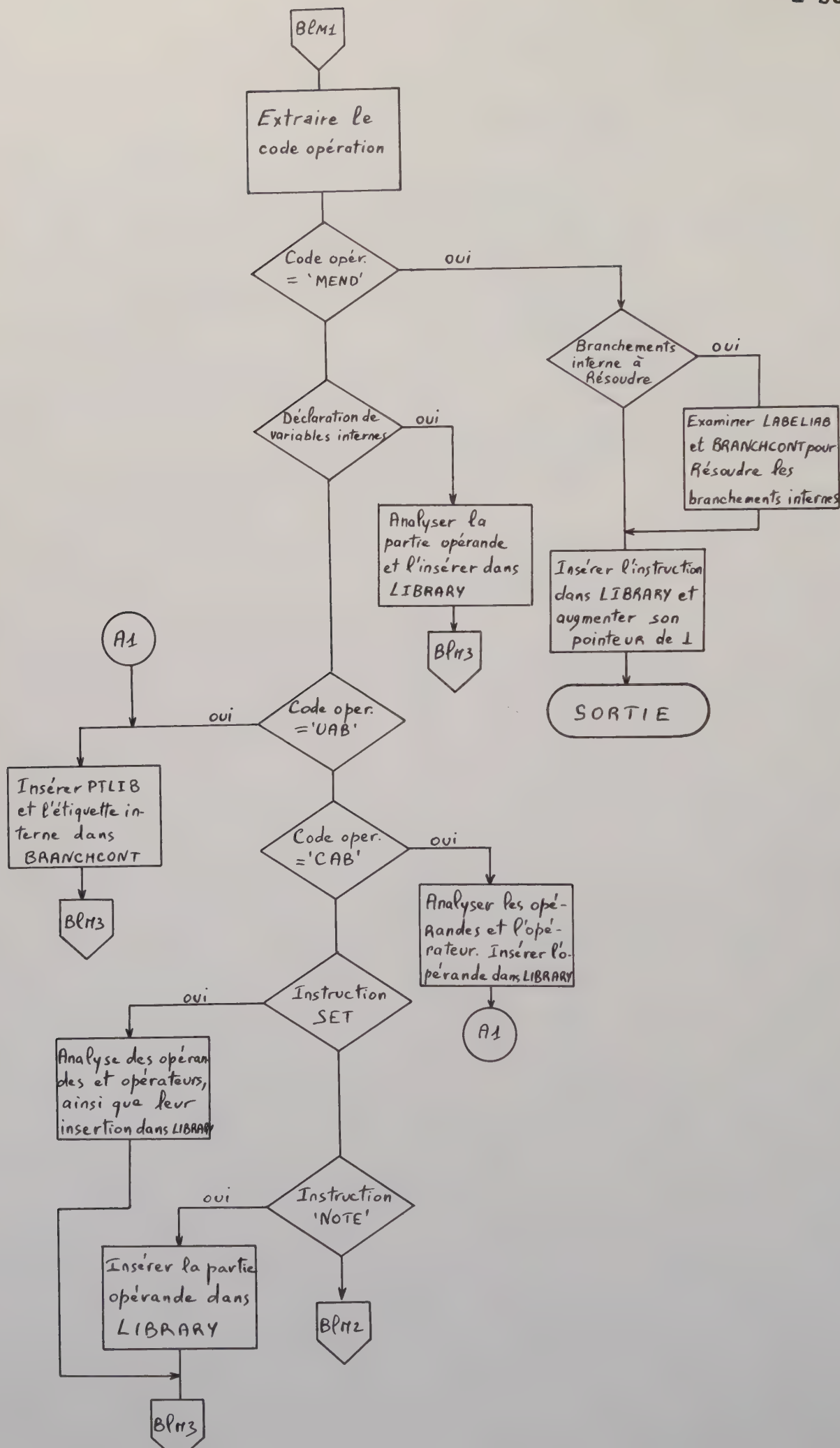
B \$PAR1 , - - COMMENTAIRE - -

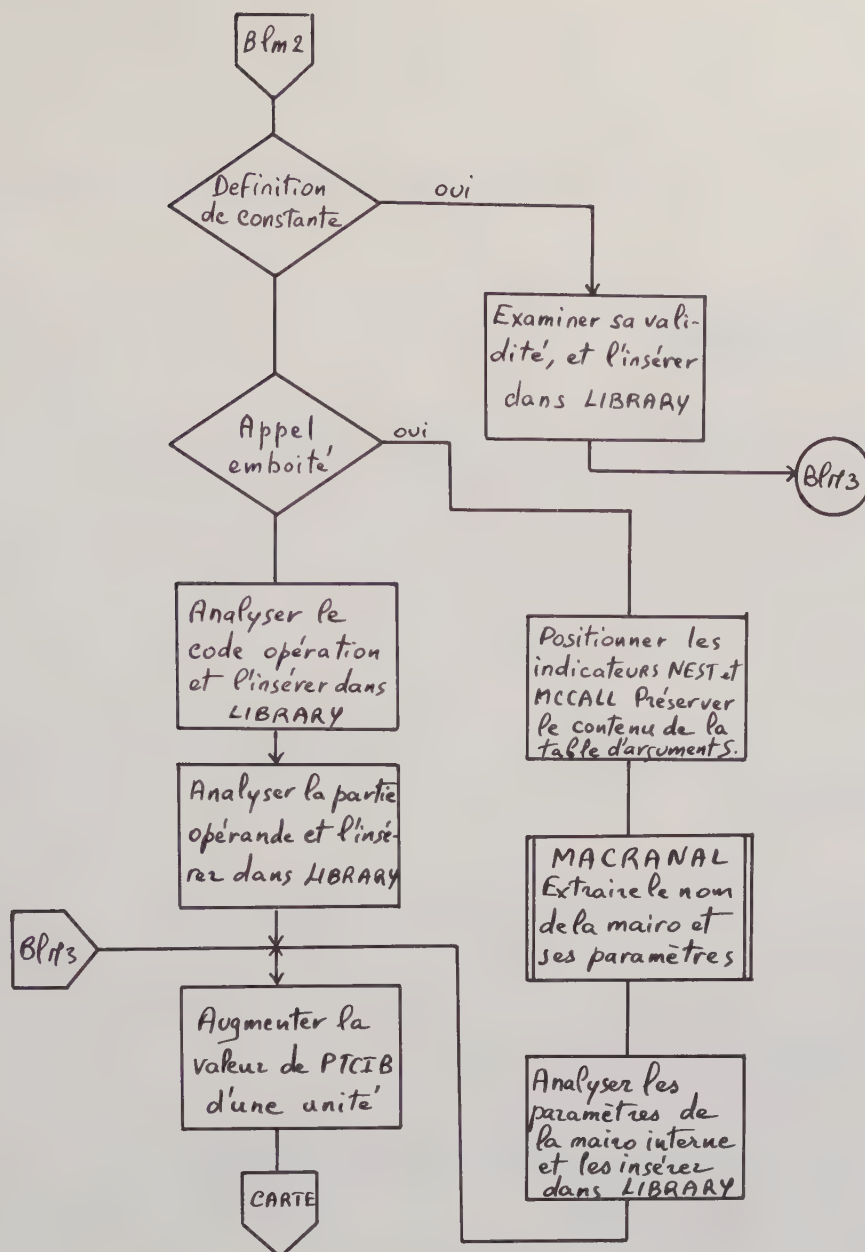
Le premier commentaire est simplement ignoré.

Le deuxième est inséré dans la bibliothèque et reproduit à chaque fois que l'instruction de chargement est générée.

2-10 Organigramme Général pour MACLIB







3 - CALLLIB

3-1 Introduction

CALLLIB est appelée par MACGEN à chaque fois qu'un appel de macro est détecté. Elle suppose que le corps de la macro est en mémoire centrale et prend PTLIB comme le pointeur de l'entrée en bibliothèque (LIBRARY) qui contient la première instruction de la macro correspondante.

A ce stade, aucune analyse syntaxique n'est effectuée, car chaque entrée dans la bibliothèque est censé avoir été mise correctement par MACLIB

Lorsque le caractère "\$" est détecté au début d'un identificateur, le caractère qui suit est examiné. S'il s'agit d'une lettre, l'identificateur se réfère à une variable interne. Autrement c'est une référence à un paramètre formel.

Initialement, CALLLIB transfère les parties étiquette et code operation (LABEL et CODE) de l'entrée correspondante en bibliothèque et les place l'une après l'autre dans une région de travail (AUXOP2) longue de 15 caractères.

En AUXOP2 elles sont examinées pour qu'on puisse détecter des paramètre formels, des variables internes ou des opérateurs de concaténation. Si des paramètres formels sont détectés, l'équivalent numérique des deux caractères qui représentent leur ordre dans la liste des paramètres, est utilisé comme index pour chercher sa valeur effective dans la table d'arguments (ARGUM).

Si une entrée de la forme

\$01 B \$02+1

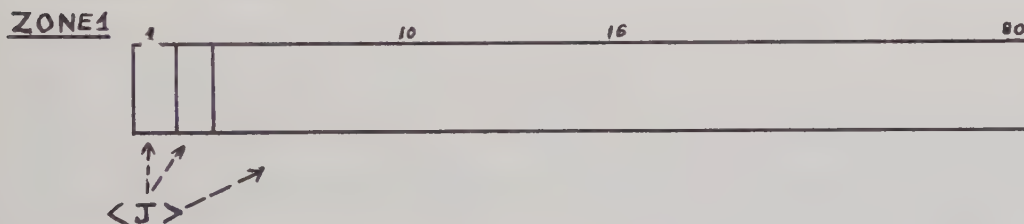
existe en bibliothèque, la valeur 1 est utilisée pour accéder à

l'élément ARGUM (1,NIVD) qui est l'étiquette effective.

La valeur de la deuxième dimension NIVD indique le niveau d'emboîtement. Lorsqu'une macro externe est appelée, NIVD est d'abord positionné à 1. Si un appel emboîté est détecté, sa valeur est augmentée de 1 pour indiquer un niveau d'emboîtement de 2. Dans le cas où la macro interne génère elle aussi un appel emboîté, NIVD sera augmenté de nouveau.

Le mécanisme conçu pour la gestion des appels emboîtés sera décrit en détail dans le paragraphe 3.10.

CALLLIB cherche la valeur effective du paramètre formel et la place à partir de la position J dans une région de 80 caractères (ZONE1).



Le pointeur J évolue pour indiquer la position d'insertion correcte en ZONE1.

Lorsque la génération d'une instruction se termine, le caractère "+" est inséré en position 70 et le bloc entier (ZONE1) est écrit sur disque.

Les étiquettes sont insérées en commençant à la position 1, les codes opération en 10 et les opérandes à partir de 16. Dans les rares cas où cela s'avère impossible, J évolue de façon à assurer une insertion correcte comme dans le cas de concatenations à effectuer.

La valeur du pointeur PTOUT, évolue de 1 à chaque fois qu'une instruction est générée.

3-2 Variables internes

Si la présence d'une variable interne est détectée, une procédure interne SEARCHINT est appelée pour examiner les tables qui gardent les noms et valeurs des variables internes.

Ces tables sont déclarées sous les noms :

LOCALARITM	Variables arithmétiques locales
LOCALCHAR	Variables caractères locales
GLOARIT	Variables arithmétiques globales
GLOCHAR	Variables caractères globales.

Le nombre de variables locales, pour chaque cas, est limité à 30. Ce nombre est suffisant car lorsqu'une macro se termine, tout l'espace occupé par ces variables est libéré. En fait, 30 est le nombre maximum de variables locales de chaque type, qui peuvent être définies par macro.

Pour les variables globales, le nombre est limité à 110 pour les arithmétiques et à 100 pour les variables caractères. Toutes ces dimensions peuvent être facilement altérées si un nombre plus grand est requis.

Le format de ces tables est

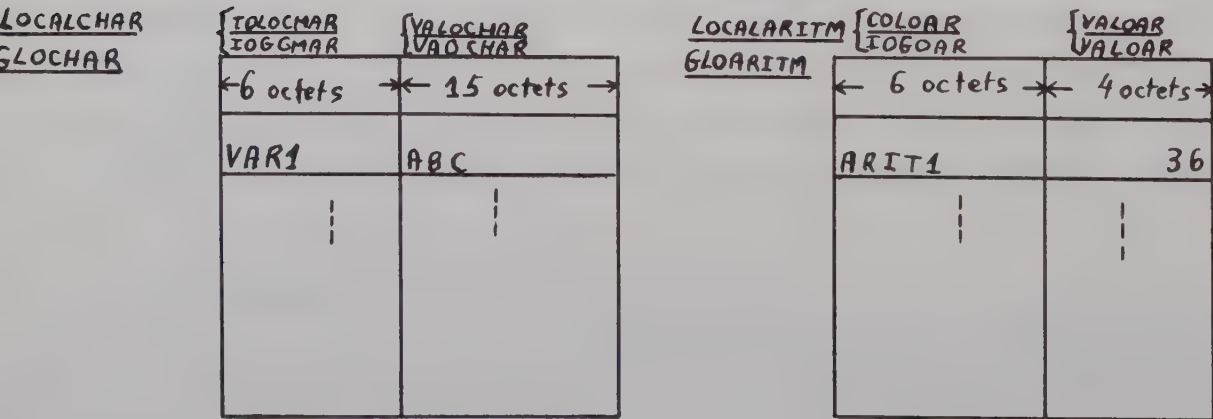


fig. 10

L'organisation des tables locales et globales est identique. Elles ne diffèrent que par le nombre d'entrées permises pour chaque cas.

Pour chaque entrée dans toutes les tables il y a une partie qui contient le nom de la variable interne correspondante, justifiée à gauche et remplie avec des blancs à droite si nécessaire. L'autre partie contient la valeur associée à cette variable interne. Pour les variables caractères, 15 octets sont utilisés pour représenter leurs valeurs. Dans le cas des arithmétiques, 32 bits (le premier étant le signe) représentent leurs valeurs numériques.

Lorsqu'une variable interne est détectée en bibliothèque, elle est placée dans une zone de 6 caractères (INTERNAL). La procédure SEARCHINT est appelée pour examiner les tables de variables internes en cherchant l'identificateur représenté dans INTERNAL. S'il est trouvé, SEARCHINT place la valeur associée à cette variable interne soit en INTARIT (variable arithmétique) soit en INTCHAR (variable caractère).

Les indicateurs TYPE et LOCAL sont positionnés par SEARCHINT pour indiquer la nature de la variable (arithmétique ou caractère) et son domaine d'application (local ou global).

Les variables arithmétiques sont converties à la forme caractère et placées dans les positions correctes de ZONE1. De cette manière, les variables internes peuvent être utilisées ainsi que les paramètres formels pour générer du code TAS-86.

La recherche dans les tables de variables internes est commandée par les pointeurs respectifs. Le mécanisme par lequel ces pointeurs sont mis à jour est expliqué dans le paragraphe suivant.

3-3 Déclaration de variables internes

3-31 Variables globales

Si une des instructions utilisées pour déclarer des variables globales (DCGA ou DCGC) est détectée, il faut d'abord tester si la macro qui est en train d'être analysée a été ou non appelée antérieurement. Pour le premier appel, toutes les mesures qui concernent la déclaration de variables globales sont prises. Par contre, si la macro a été antérieurement appelée, les instructions de déclarations globales doivent être simplement ignorées.

Pour les macros définies par le programmeur, le troisième champ (SWITCH) associé à chaque entrée dans le répertoire (DIRECTO) est utilisé pour indiquer si une certaine macro a été ou non antérieurement appelée. Initialement, toutes les entrées correspondant à SWITCH sont mises à zéro.

Lorsqu'une macro est appelée, la valeur correspondante de SWITCH est testée.

DIRECTO

NAME	ADRESSE	SWITCH
nom de la macro	pointeurs pour la bibliothèque	SWITCH

fig. 11

La valeur zéro indique que la macro n'a jamais été appelée.

La valeur de SWITCH est associée à une variable auxiliaire SWITCH1. Ensuite, l'entrée correspondante de SWITCH est positionnée à 1. En déclarant une variable globale, la valeur de SWITCH1 indique si l'instruction doit être ignorée ou non.

Dans le cas de macros systèmes où il n'y a pas un répertoire en mémoire centrale, on introduit dans une zone de 210 caractères initialisée avec des blancs (ACTMAC), les noms de macros système lorsqu'elles sont appelées pour la première fois. La présence ou non d'un certain nom de macro en ACTMAC servira à positionner SWITCH1 correctement.

3-32 Variables locales

Avant la déclaration d'une variable interne quelconque, la procédure SEARCHINT est appelée avec l'indicateur FIND positionné à 1. Les tables sont examinées afin de vérifier si elles contiennent déjà la variable interne qui va être déclarée. Sinon, un retour normal a lieu et la variable en question est insérée dans la table correspondante. Dans le cas où le nom existe déjà, deux possibilités se présentent:

- a) le nom existe dans une table de la même nature que la variable à être déclarée. Par exemple, un identificateur LCHAR, qui doit être déclaré caractère local, se trouve dans LOCALCHAR. Dans ce cas, un simple avertissement de double déclaration est imprimé.
- b) le nom existe dans une table dont la nature diffère de celle de la variable. Une erreur de déclaration double non compatible est alors détectée.

Dans les cas normaux, l'identificateur est simplement inséré dans la table correspondante avec l'initialisation requise.

DCLA COUNT
DCGC CHAR

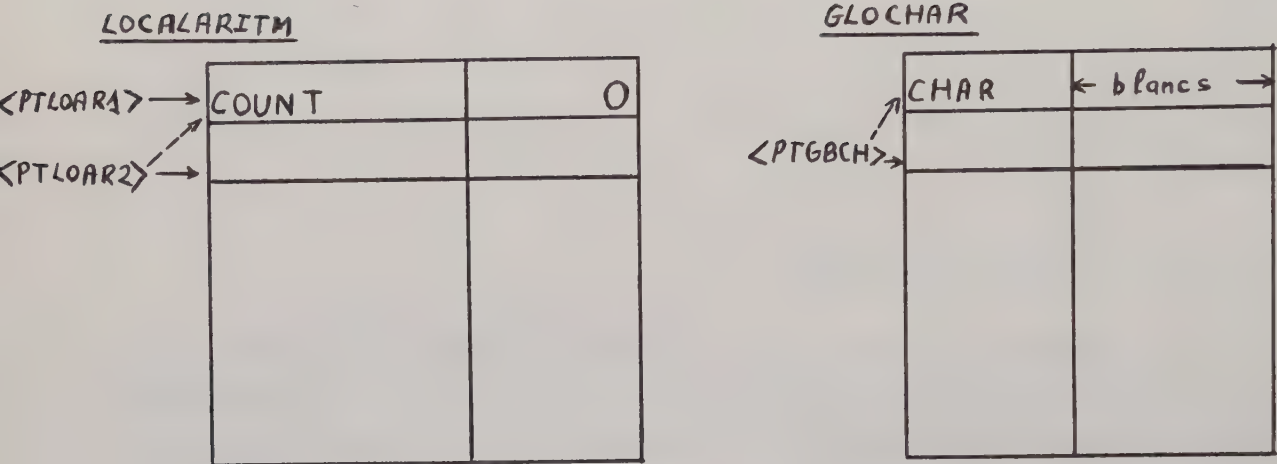


fig. 12

Les tables globales sont gérées par leurs pointeurs PTGBAR et PTGBCH. A chaque fois qu'une nouvelle variable est détectée, ils sont augmentés de 1. Leurs valeurs limitent la recherche d'un identificateur dans la table correspondante.

Les tables locales, requièrent deux pointeurs pour gérer leur mécanisme. PTLOAR1 et PTLOCHAR1 indiquent la première entrée disponible dans chaque table. Lorsque la macro est appelée, ils

sont initialisés à 1. A chaque fois qu'une variable locale est déclarée, PTLOAR2 ou PTLOCHAR2 sont augmentés de 1.

Dès que l'on accède à une variable locale, la recherche débute à l'entrée pointée par PTLOAR1 ou PTLOCHAR1, et termine à celle spécifiée par PTLOAR2 ou PTLOCHAR2.

Les premiers pointeurs servent à contrôler le domaine d'existence des variables locales dans l'éventualité des appels emboîtés. Les variables locales déclarées dans une macro externe deviennent inaccessibles dans une macro interne ; cependant, leurs valeurs ne sont pas perdues. Les valeurs de pointeurs PTLOCHAR1 et PTLOAR1 sont empilées et de nouvelles valeurs leurs sont affectées.

Ces mécanismes seront décrits en détail dans le paragraphe consacré aux appels emboîtés.

Dans le cas ordinaire (inexistence d'appels emboîtés), PTLOAR1 et PTLOCHAR1 sont toujours égaux à 1 et les tables seront gérées par PTLOAR2 et PTLOCHAR2 de la même manière que les tables globales.

3-4 Les instructions SETA et SETC

Ces instructions interprètent la partie opérande de l'entrée en bibliothèque correspondante de façon à exécuter les opérations spécifiées. Après l'exécution des opérations indiquées, le résultat est affecté à la variable objet.

Pour l'instruction SETA, les valeurs effectives des opérandes sont placées dans les variables arithmétiques INTARIT1 et INTARIT2 (s'il en existe une seconde).

Le résultat de l'opération arithmétique reste dans INTARIT1 qui est ensuite affecté à la variable objet.

L'indicateur TYPEAUX précise le domaine d'application (local ou global) pour la variable objet et PTAUX indique l'ordre de l'entrée dans une des tables qui la contient.

Ces deux valeurs s'obtiennent par l'appel de SEARCHINT au moment de la détection de la variable objet.

L'instruction SETC fonctionne d'une façon très analogue. Le résultat des opérations reste dans une variable caractère (INTCHAR) avant d'être affecté à la variable objet. TYPEAUX et PTAUX jouent de la même manière que pour les instructions SETA.

L'opération MOVE est analysée, ses trois opérandes sont placées dans MOVEOP (tableau de trois éléments arithmétiques) et la chaîne de caractères d'où le transfert aura lieu est placée dans INTCHAR.

En utilisant la fonction SUBSTR (fonction standard du langage PL/1), le transfert est exécuté ayant comme résultat une partie ou la totalité du contenu de la chaîne source placée dans la variable objet.

3-5 L'instruction UAB

L'exécution de l'instruction UAB est très simple.

La valeur du déplacement relatif représenté sous la forme caractère à la fin de la partie opérande (OPERAND), de l'entrée de la bibliothèque en train d'être analysée, est convertie en une valeur numérique qui vient s'ajouter à la valeur courante de PTLIB.

Ex : UAB .END -32

Nous supposons que l'instruction nommée END se situe en bibliothèque 32 entrées avant l'instruction UAB. Dans ce cas, la valeur de PTLIB est diminuée de 32.

3-6 L'instruction CAB

Les deux opérandes et l'opérateur de l'instruction CAB sont analysés pour déterminer le type de comparaison qui sera faite entre les deux opérandes.

Si l'un des opérandes est une variable interne arithmétique, l'autre opérande est automatiquement converti en une forme numérique (si cela n'est pas possible, une erreur de conversion sera produite).

Ex : CAB (\$\$COUNT GT \$VAR).END

En supposant que COUNT a été déclarée comme une variable arithmétique la valeur effective du paramètre formel VAR sera convertie dans un nombre binaire ce qui permettra d'effectuer la comparaison.

Toutes les différentes possibilités pour l'opérateur et les opérandes sont examinées et selon le cas l'action correcte est prise.

Quelques opérateurs sont équivalents. Par exemple :

NL (pas plus petit) = GE (plus grand ou égal).

NG (pas plus grand) = LE (plus petit ou égal).

Dans ces cas, les actions prises sont exactement les mêmes. Dans d'autres cas, l'action déterminée par un opérateur

représente la négation d'une autre.

LT (plus petit que) $\neg$ = GE (plus grand ou égal)

LE (plus petit ou égal) $\neg$ = GT (plus grand)

NE (non égal) $\neg$ = EQ (égal)

Ces relations peuvent être utilisées pour économiser beaucoup de codage car les parties du programme qui analysent les différentes conditions deviennent communes selon les équivalences ou les négations. Pour les dernières, un indicateur (REVERT) est positionné à 1. Si par exemple, l'opérateur NE est détecté, l'expression s'analyse de la même façon que pour l'opérateur EQ. Au moment d'effectuer le branchement ou non, selon le résultat du test, REVERT est testé. S'il est égal à 1, l'action déterminée par EQ est simplement inversée.

Les indicateurs I, INT et IND sont positionnés afin de préciser le type d'opérateur et d'opérandes de la façon suivante :

I =	{	1 $\longrightarrow$	opérateur EQ	INT =	{	0	1er opérande - paramètre formel
		2 $\longrightarrow$	opérateur NE			1	1er opérande - variable interne
		3 $\longrightarrow$	opérateur GT			2	1er opérande - constante
		4 $\longrightarrow$	opérateur GE				
		5 $\longrightarrow$	opérateur LT				
		6 $\longrightarrow$	opérateur LE				
		7 $\longrightarrow$	opérateur NG				
		8 $\longrightarrow$	opérateur NL				
IND =	{	0 $\longrightarrow$	2e opérande - paramètre formel				
		1 $\longrightarrow$	2e opérande - variable interne				
		2 $\longrightarrow$	2e opérande - constante				
		3 $\longrightarrow$	2e opérande - caractère spécial				

Ces trois indicateurs (I, INT, IND) précisent exactement le genre de test qui doit être soumis aux opérandes et déterminent ainsi un point de branchement dans le programme où l'analyse requise sera effectuée.

Les variables auxiliaires AUXOP1 et AUXOP2 sont utilisées pour garder les valeurs effectives des deux opérandes sous forme caractère et INTARIT1 et INTARIT2 effectuent la même fonction dans le cas où leurs valeurs doivent être converties en une forme arithmétique.

3-7 L'instruction NOP

Son exécution est immédiate. La valeur de PTLIB est simplement augmentée de 1.

3-8 L'instruction NOTE

La partie opérande de l'entrée correspondante en bibliothèque est imprimée précédée de "****WARNING****".

3-9 L'instruction NPUN

Elle provoque l'affectation de la valeur 2 à l'indicateur NOPUNCH, ce qui empêchera le programme généré d'être perforé sur cartes.

3-10 Appels emboîtés

3-101 Préservation de données

Lorsque pendant la génération d'une macro un appel

emboîté est détecté, la première mesure à prendre est de préserver toutes les informations importantes concernant la macro qui génère l'appel. Ceci fait, le contrôle passe alors à la macro appelée.

Plusieurs piles sont utilisées pour garder ces informations. Leurs noms et fonctions sont décrits ci-dessous :

a) NESTSCOPE

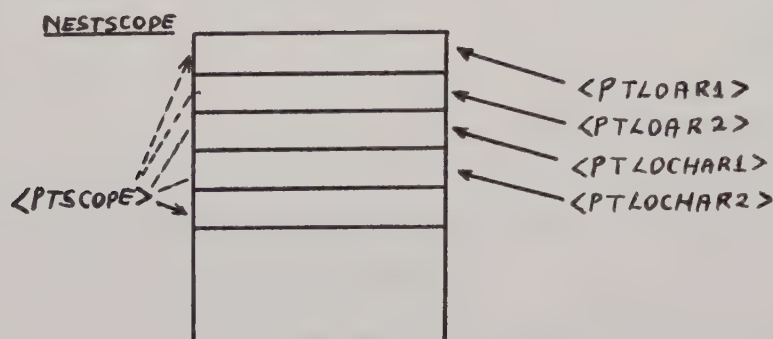


fig. 13

Cette pile est utilisée pour garder les valeurs de tous les pointeurs en relation avec les variables internes locales.

Comme nous le savons, les variables locales déclarées dans une macro externe ne sont pas accessibles durant la génération de la macro interne quoique leurs valeurs ne se perdent pas.

Nous décrirons le mécanisme dans le cas de variables arithmétiques locales puisque pour les variables caractères il en est exactement de même.

Supposons que, pendant la génération de la macro externe 3 variables arithmétiques locales aient été déclarées.

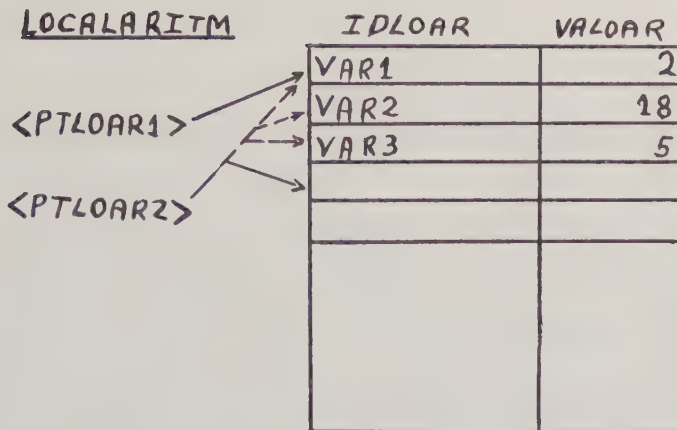


fig. 14

Si un appel emboîté est détecté, les valeurs de PTLOAR1 et PTLOAR2 sont introduites dans NESTSCOPE. La valeur de PTLOAR2 qui indique la prochaine entrée libre dans LOCALARITM est alors affectée à PTLOAR1.

Cette opération commande le domaine d'application pour les variables locales. En ce qui concerne la macro interne, la première entrée libre pour insérer des variables locales qui soient éventuellement déclarées dans LOCALARITM, sera donnée par la nouvelle valeur de PTLOAR1. Les variables déclarées antérieurement deviennent simplement inaccessibles.

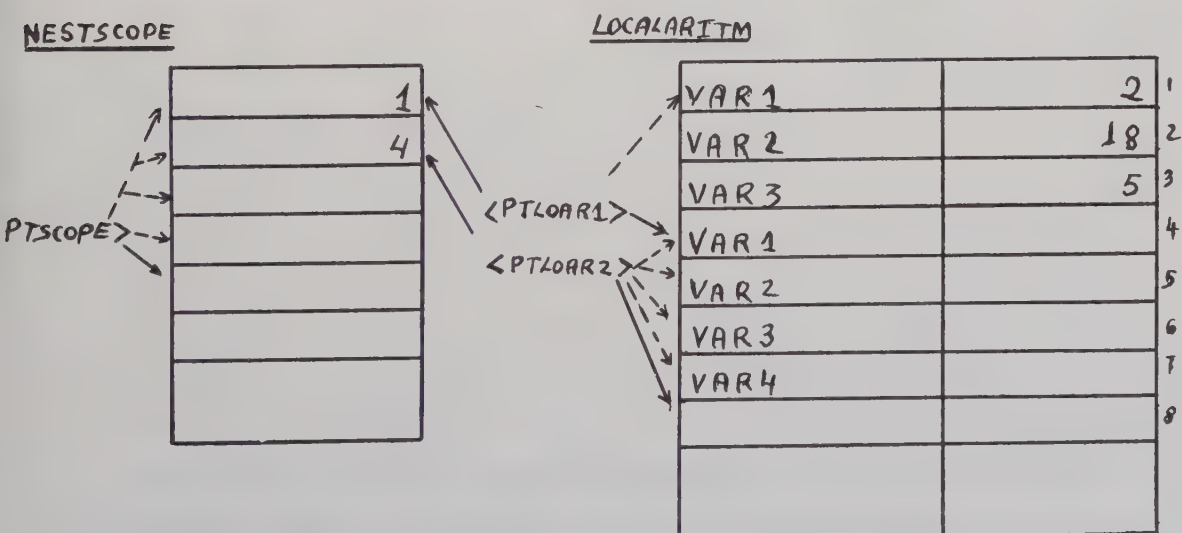


fig. 15

Dans l'exemple de la figure 15, nous supposons que durant la génération de la macro externe, trois variables internes avaient été déclarées comme le montre la figure 14. La macro interne déclare alors 4 nouvelles variables locales qui ne dépendent pas du tout des précédentes bien qu'elles puissent avoir le même nom.

La recherche des variables arithmétiques locales dans la macro interne sera toujours limitée entre les entrées 4 et 8 de LOCALARITHM. Si la macro interne ne génère pas elle-même un appel emboîté, lorsqu'elle terminera, les conditions précédentes sont rétablies (comme nous le verrons dans la description de l'instruction MEND) grâce aux informations stockées dans NESTSCOPE. Cependant, si la macro interne génère un autre appel emboîté, les nouvelles valeurs de PTLOAR1 et PTLOAR2 seront empilées en NESTSCOPE et le même mécanisme sera répété.

b) CONDSTACK

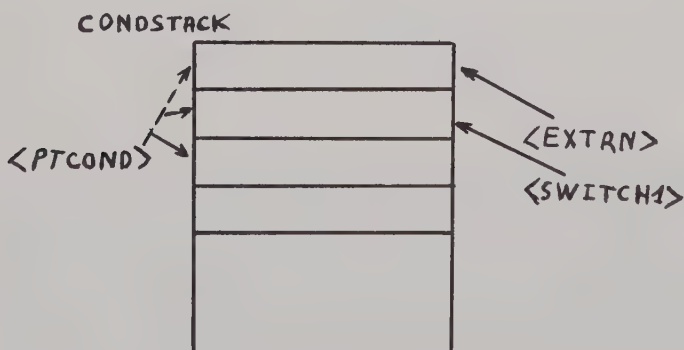


fig. 16

CONDSTACK est employée pour garder des informations concernant quelques conditions de la macro externe. Ces informations

sont représentées par les deux indicateurs SWITCH1 et EXTRN.

SWITCH1, comme nous l'avons vu, indique si la macro en train d'être générée a déjà été ou non antérieurement appelée. EXTRN précise s'il s'agit ou non d'une macro système. Ces deux indicateurs doivent être préservés car ils seront repositionnés pour la macro interne. Ils sont rétablis de la même façon que les pointeurs PTLOAR1 etc, à la fin de la génération de la macro interne sauf si un autre appel est généré ce qui produira un empilement d'autres informations et ainsi de suite.

c) NESTNAME et NESTADDR

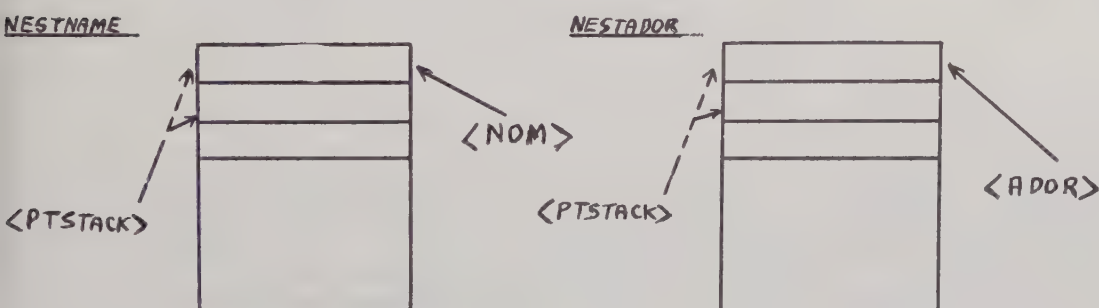


fig. 17

Elles sont employées pour préserver les informations concernant le nom et l'adresse initiale de la macro en train d'être générée. NOM est une zone de 6 caractères qui contient le nom de la macro et ADDR indique l'entrée en bibliothèque qui précède immédiatement la première instruction de la macro en train d'être générée. Ces données sont requises car en cas d'erreurs détectées pendant la génération, NOM et ADDR sont utilisés pour préciser le nom de la macro et l'ordre de l'instruction responsable de l'erreur. Comme ils sont automatiquement changés pour la macro interne, ils doivent être empilés pour être rétablis au moment nécessaire.

d) NESTSTACK

C'est une pile de retour qui garde l'adresse de retour pour la macro externe. Lorsqu'un appel emboîté est détecté, la valeur de PTLIB correspondant à l'instruction qui suit immédiatement celle qui a généré l'appel, est introduite dans NESTSTACK. Quand la macro interne arrive à sa fin, le pointeur sera extrait de NESTSTACK pour la prochaine instruction à être générée dans la macro externe.

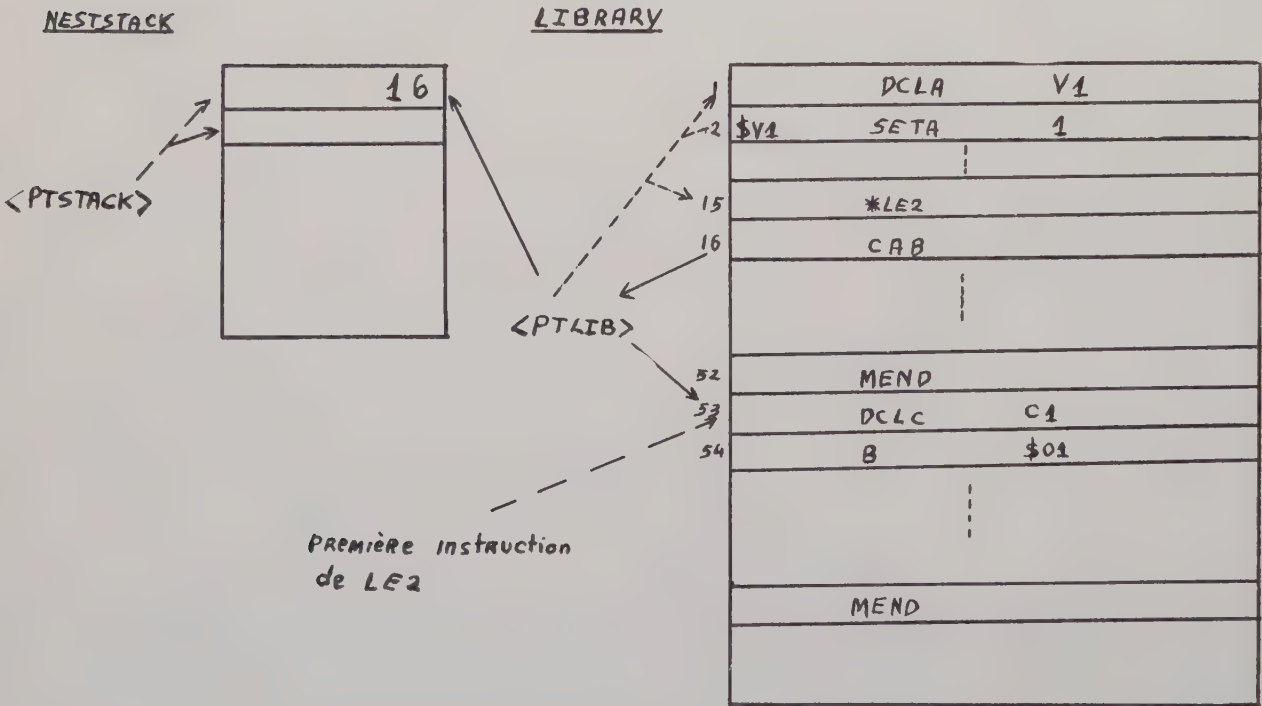


fig. 18

Si un appel emboîté se détecte à l'entrée 15 de LIBRARY par exemple, CALLLIB déterminera la valeur de PTLIB correspondant à la première instruction de la macro interne. Cependant, avant

d'affecter cette nouvelle valeur à PTLIB, la valeur 16 correspondant à l'instruction qui suit l'appel emboîté est introduite dans NESTSTACK. En retournant de la macro interne, la génération se poursuivra normalement à l'entrée 16 de LIBRARY.

Il est évident que, si la macro interne génère un appel emboîté, une nouvelle adresse de retour sera stockée dans NESTSTACK, provoquant par conséquent une descente de pile.

3-10.2 Transmission de paramètres dans les appels emboîtés

Avant de commencer la génération de la macro interne, ses paramètres effectifs doivent être placés dans les positions correctes de la table d'arguments (ARGUM).

L'indicateur NIVD est augmenté de 1 à chaque fois qu'un appel emboîté est détecté et diminué de la même quantité lorsqu'une macro arrive à sa fin. Au moment d'accéder aux paramètres effectifs, CALLLIB se réfère à l'élément ARGUM (PNT, NIVD). La valeur de PNT est extraite de la bibliothèque et indique l'ordre de l'élément dans la liste de paramètres. Ainsi, la valeur de NIVD précise la colonne de ARGUM dans laquelle ont été placés les paramètres effectifs. A chaque colonne on fait correspondre un niveau d'emboîtement.

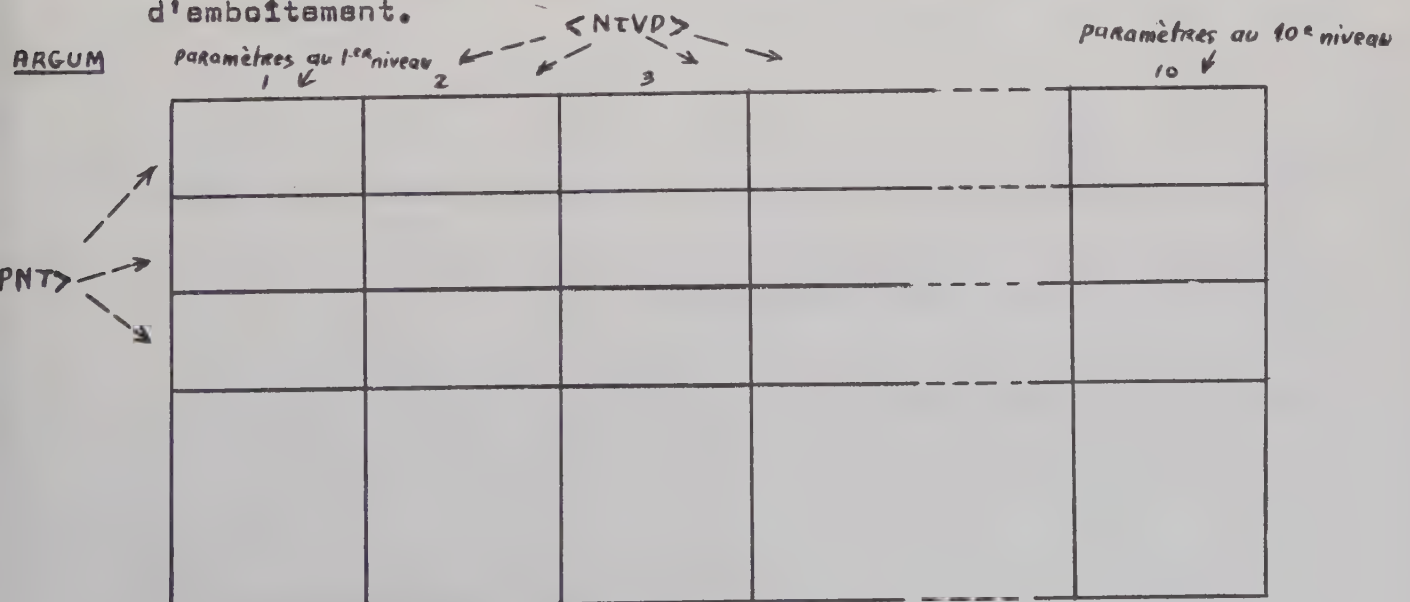


fig. 19

Lorsqu'un appel de macro est détecté, MACRANAL place dans la première colonne de ARGUM les paramètres effectifs de la macro externe. Si au cours de la génération un appel emboîté est détecté, CALLLIB analyse la liste des paramètres de la macro interne qui a été insérée dans la bibliothèque par MACLIB pendant la définition.

L'appel de la macro interne peut spécifier des paramètres formels (définis dans la macro externe) qui occupent une certaine position dans sa liste. Cette position peut ne pas être la même que celle qu'ils occupent dans la liste de paramètres de la macro externe.

Une table arithmétique (NESTMATRIX) garde les informations concernant la transmission de paramètres. L'algorithme appliqué se trouve mieux décrit sur un exemple complet.

Nous supposons une certaine macro NIV1 qui génère un appel emboîté à NIV2 définie ainsi :

```

                MACRO
$LAB   =      NIV1  $PAR1,$PAR2,$PAR3,$PAR4
        :
        :
$PAR1  =  *NIV2  $LAB,V1,V2,$PAR1
        :
        MEND

```

La macro NIV2 génère elle-même un appel emboîté à NIV3

```

                MACRO
$LAB   = NIV2  $PAR1,$PAR2,$PAR3,$PAR4
        :
        :
$PAR3  = *NIV3  $PAR4,ZONE
        :
        MEND

```

En supposant que NIV3 ne génère pas un autre appel, le niveau d'emboîtement est alors limité à 3.

Si un appel du genre :

NAME= *NIV1 TABLE,VAR,1

est détecté, MACGEN active MACRANAL pour extraire les paramètres effectifs et les placer dans ARGUM.

<NIV0>

ARGUM

	1	2	3	10
0	NAME			
1	TABLE			
2	VAR			
3	1			

fig. 20

MACRANAL remplit la première colonne de ARGUM avec les paramètres effectifs de NIV1. L'instruction en bibliothèque correspondante à l'appel emboîté à NIV2 a la forme :

\$01	*NIV2	\$00, V1, V2, \$01, /
------	-------	-----------------------

En analysant cette entrée, CALLLIB place directement dans ARGUM les paramètres de NIV2 qui sont déjà effectifs.

Les valeurs numériques représentant l'ordre des paramètres formels (cet ordre correspond à la liste de paramètres de la macro externe), sont placées dans l'entrée de NESTMATRIX qui correspond à l'ordre de ce même paramètre dans la liste de la macro interne. Si le paramètre est déjà effectif dans la liste interne, la valeur 999 est introduite dans NESTMATRIX.

Les valeurs introduites en NESTMATRIX, selon cet exemple, sont :

NESTMATRIX

0	1
1	0
2	999
3	999
4	1
	⋮

La valeur 1 indique que l'étiquette de l'instruction d'appel emboîté, c'est-à-dire le paramètre d'ordre zéro de la macro interne, est donnée par le premier paramètre de la macro qui génère l'appel.

fig. 21

Les entrées dans NESTMATRIX sont alors utilisées pour effectuer la correspondance correcte entre les paramètres

ARGUM

0	NAME	TABLE			
1	TABLE	NAME			
2	VAR	V1			
3	1	V2			
4		TABLE			

fig. 22

Lorsqu'une entrée a la valeur 999 en NESTMATRIX, la valeur du paramètre correspondant est cherchée directement dans la liste de la macro interne.

L'appel emboîté de NIV3 est représenté en bibliothèque par :

\$03	*NIV3	\$04, ZONE, /
------	-------	---------------

fig. 23

Les entrées générées en NESTMATRIX et la transmission de paramètres dans ARGUM qui en découle sont :

NESTMATRIX

0	3
1	4
2	999
3	
4	

ARGUM

0	NAME	TABLE	V2
1	TABLE	NAME	TABLE
2	VAR	V1	ZONE
3	1	V2	
4		TABLE	

Diagram showing parameter transmission from NESTMATRIX to ARGUM:

- NESTMATRIX index 0 points to ARGUM index 0 (NAME).
- NESTMATRIX index 1 points to ARGUM index 1 (TABLE).
- NESTMATRIX index 2 (999) points to ARGUM index 2 (VAR).
- ARGUM index 2 (VAR) points to ARGUM index 3 (1).
- ARGUM index 3 (1) points to ARGUM index 4 (TABLE).

Comme la valeur de NIVD augmente ou diminue convenablement, CALLLIB accèdera toujours aux arguments effectifs de la macro en train d'être générée dans les positions correctes de ARGUM.

Cet exemple démontre bien la façon par laquelle les paramètres peuvent être transmis entre des macros en différents niveaux. Après avoir complété la transmission de paramètres, CALLLIB passe à générer les instructions faisant partie de la macro interne.

3-11 L'instruction MEND

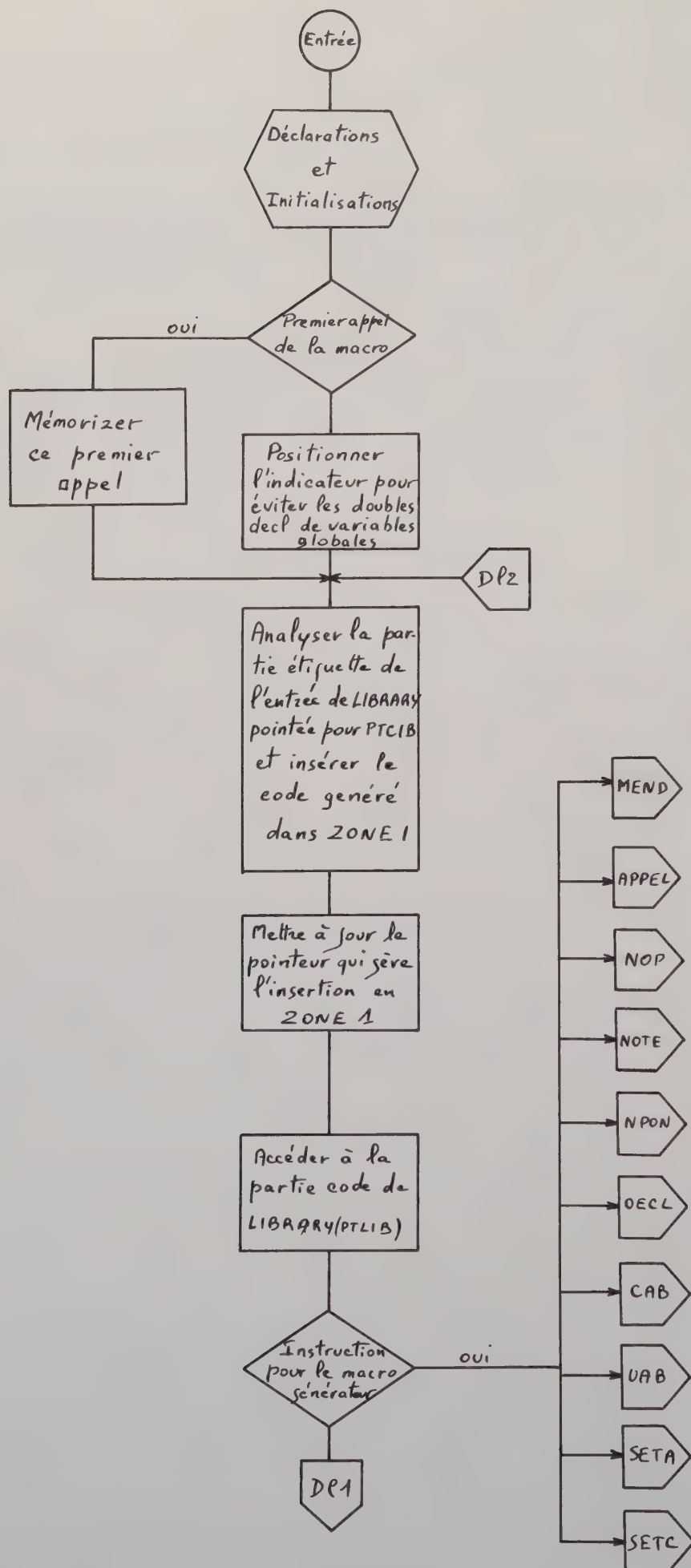
Lorsque l'on détecte l'instruction MEND, ce qui indique la fin de la macro correspondante, plusieurs mesures sont prises pour libérer de l'espace en mémoire centrale (macros système) et pour rétablir des conditions primitives (dans les cas des appels emboîtés).

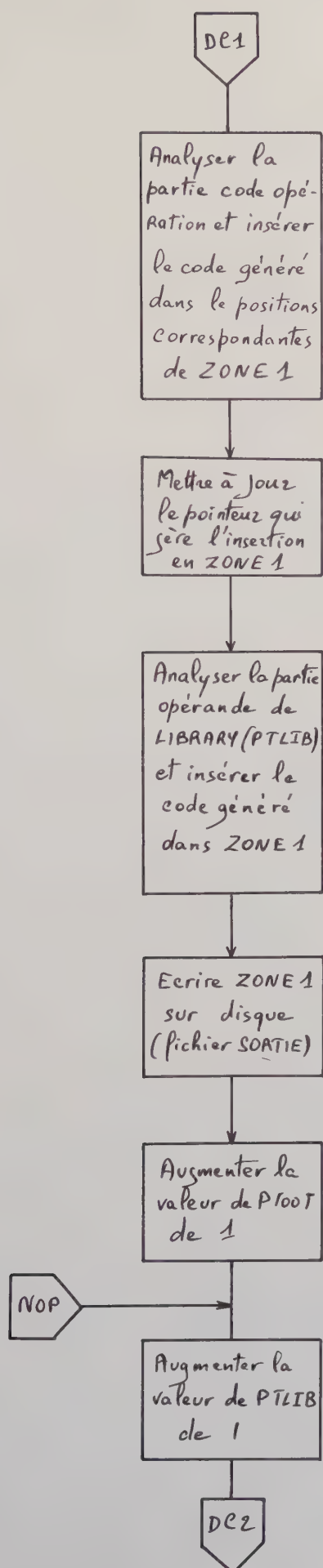
Les actions entreprises sont :

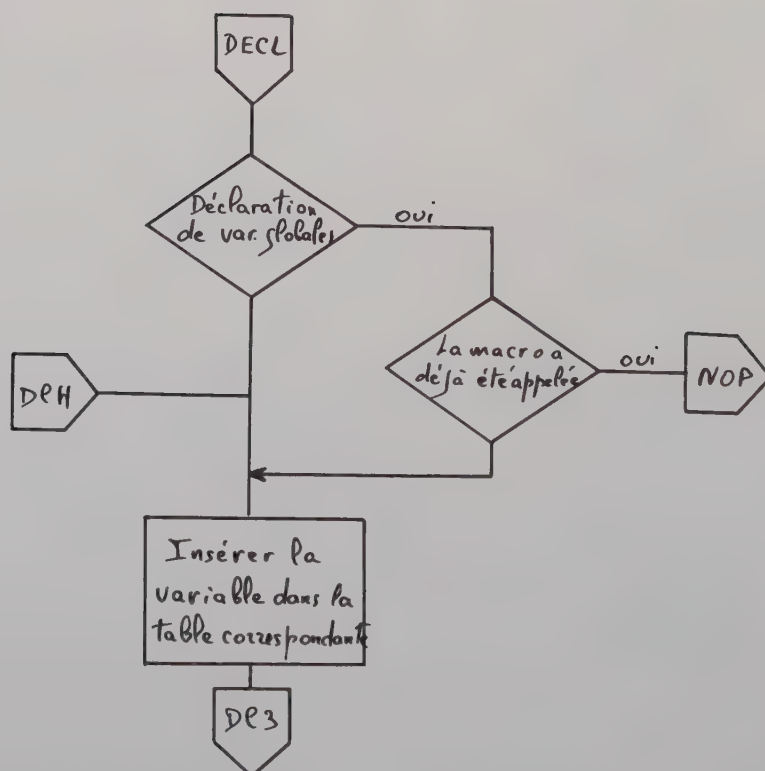
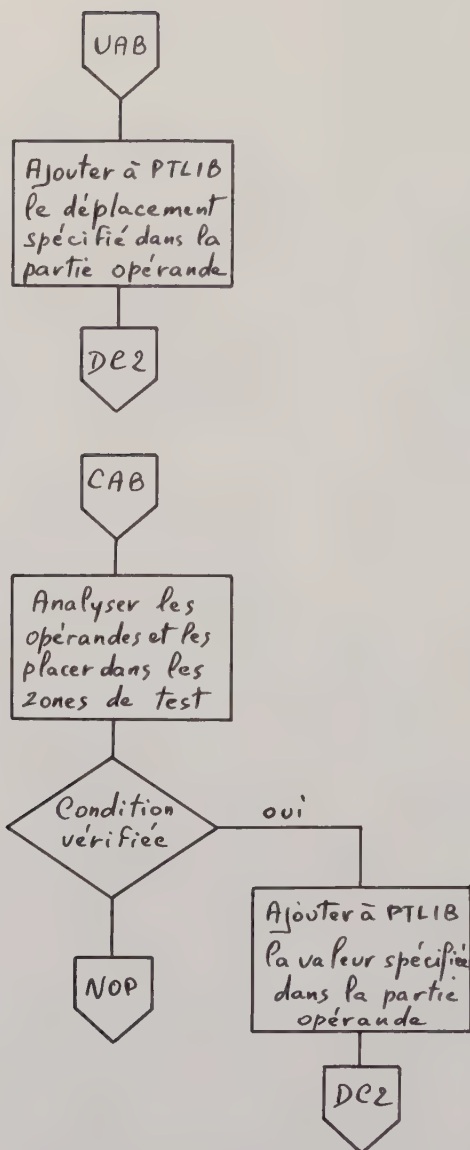
- a) Si la génération se réfère à une macro système, les entrées qu'elle occupe en bibliothèque (LIBRARY) sont libérées en effectuant une montée de pile de DYNCORE (pile qui contrôle l'allocation dynamique de la mémoire centrale). Cela se produit en diminuant son pointeur (PTDYN) de 1.
- b) Libération des entrées permises dans les tables de variables locales.
- c) Si $NIVD > 1$, ce qui signifie que la macro générée a été activée par un appel emboîté, toutes les informations préservées dans les différentes piles décrites en 3-101, sont utilisées pour rétablir les conditions qui concernent la macro qui a produit l'appel emboîté. Finalement, la valeur de NIVD est diminuée de 1 avant de

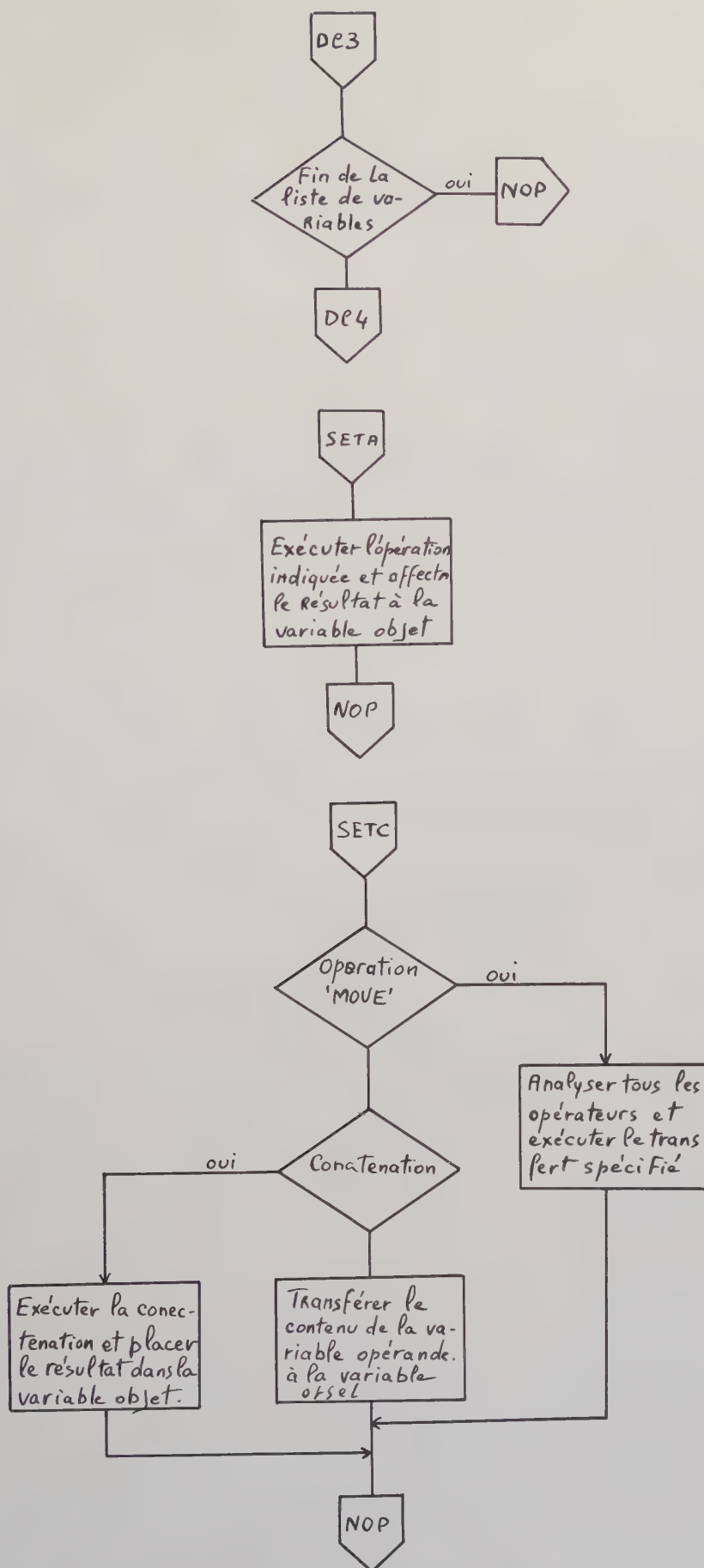
poursuivre la génération à l'adresse extraite de la pile de retour (NESTSTACK).

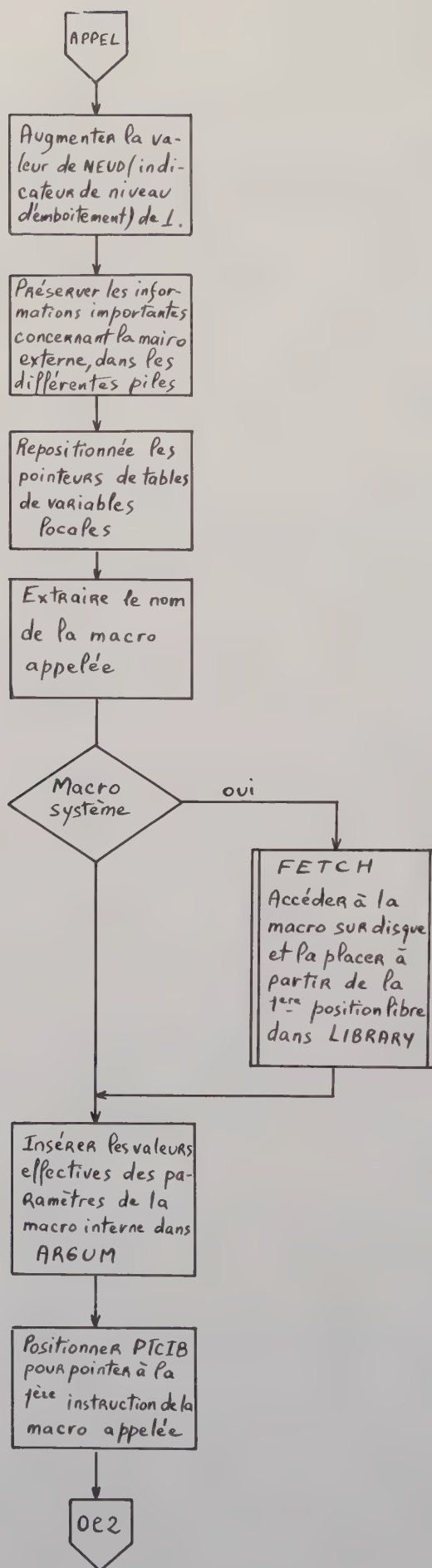
3-12 Organigramme général

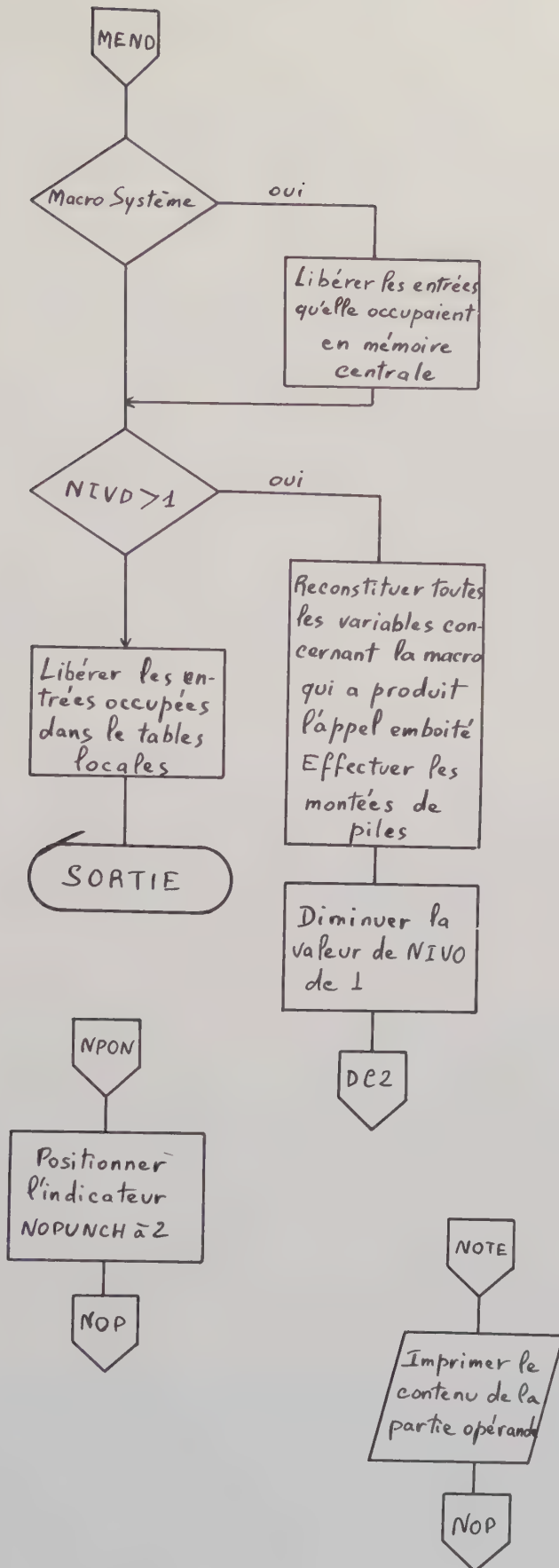












4 - PRINTIT et PRINTER

Ces modules sont responsables pour des services standards et ne présentent aucune particularité dans leur logique. PRINTIT imprime des titres généraux, ainsi que la liste des options utilisées. Les valeurs des différents indicateurs précisent pour chaque cas l'option correspondante en usage.

PRINTER lit le fichier SORTIE qui se trouve sur disque à la fin de la génération, et place le programme traduit sur les périphériques demandés (imprimante, bande magnétique, cartes perforées etc). Pour cela, il suffit de les définir par des cartes de contrôle ("data definition cards") convenables.

5 - CREATE

Ce module écrit en assembleur, est activé à chaque fois qu'une définition de macro doit être modifiée ou ajoutée à la bibliothèque système sur disque (options 'CREATE' ou 'UPDATE'). CREATE reçoit comme paramètres les variables :

a) LIBRARY

Bibliothèque en mémoire centrale qui contient le corps de la macro à écrire sur disque, après avoir été traité par MACLIB.

b) PTLIB

Pointeur qui indique le nombre de blocs de LIBRARY qui devront être écrits sur disque. En fait, la valeur de PTLIB indique la fin du corps de la macro.

c) NOM1 -

Cette variable contient le nom de la macro à ajouter ou

à remplacer dans la bibliothèque système.

d) RESULT -

La valeur de RESULT indiquera le résultat de l'opération d'écriture sur disque. Sa valeur est donnée par le code de retour laissé dans le registre 15 par la macro STOW (OS 360 macro).

e) END1 -

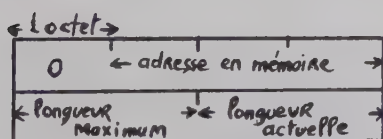
Cet indicateur est positionné à 1 la dernière fois qu'on appelle CREATE, pour provoquer la fermeture du fichier qui contient la bibliothèque système.

f) CREIND -

Il indique s'il s'agit d'une nouvelle définition à ajouter ou d'une définition déjà existante qui devra être mise à jour.

Les adresses de ces paramètres, sont passées au module écrit en assembleur, selon les conventions du compilateur PL/I. A l'exception des variables arithmétiques simples (PTLIB, RESULT, END1, CREIND) où les adresses sont passées directement, pour tous les autres cas (chaîne de caractères, bits, tableaux, structures) le compilateur PL/I passe au programme appelé écrit en assembleur les adresses des "dope vectors" correspondant à chaque variable.

Les "dope vectors" sont des zones créées par le compilateur qui contiennent des informations générales sur la variable correspondante. Pour le cas simple d'une chaîne de caractères, le "dope vector" contient 8 octets selon le format suivant :



Le premier octet est toujours zéro (dans le cas d'une chaîne de bits, il indiquera la position du premier bit de la chaîne dans l'octet correspondant).

Les 3 suivants spécifient l'adresse en mémoire de la chaîne de caractères, et les 4 derniers déterminent la longueur maximum et la longueur actuelle de la chaîne. Ces deux dernières informations sont nécessaires s'il s'agit de variables déclarées avec l'attribut VARYING. Pour les cas normaux, les deux valeurs sont toujours identiques.

Dans les cas de tableaux, structures etc., les "dope vectors" contiennent plusieurs autres informations concernant les dimensions inférieures et supérieures, description des sous éléments d'une structure et ainsi de suite. Les détails complets peuvent être obtenus dans la brochure "IBM System/360 Operating System. PL/I (F) Programmers Guide".

Ces informations sont fondamentales pour permettre au module écrit en assembleur d'accéder correctement aux données fournies par le compilateur PL/I.

Le premier bloc de la macro correspondante est écrit sur disque, dont le "track address" est déterminé par la macro NOTE (OS 360 macro), et qui est ensuite introduit dans une zone en mémoire (DIRLIST) qui contient aussi le nom de la macro.

Après l'écriture du dernier bloc, la macro STOW (OS 360 macro) placera dans le répertoire qui se trouve sur disque, les informations décrites en DIRLIST, c'est-à-dire le nom de la macro et l'adresse du bloc qui contient sa première instruction.

De cette manière, un répertoire est créé sur disque pour les macros système, qui a la même structure logique que DIRECTO,

construit en mémoire centrale pour les macros définies par le programmeur.

Dans le premier cas, le pointeur précise un endroit sur disque, tandis que le deuxième détermine une adresse en mémoire centrale.

Des informations plus complètes sur les macros OS 360, utilisées pour la gestion de la bibliothèque système sont décrites dans les brochures "Supervisor and Data Management Macro Instructions" et "Supervisor and Data Management Services" IBM System/360.

A la fin de toutes ces opérations, CREATE place le contenu du registre 15 (code de retour de la macro STOW) dans RESULT avant de passer le contrôle du système à MACGEN.

6 - FETCH

FETCH comme nous le savons, est activé par MACRANAL ou CALLLIB, à chaque fois qu'un appel à une macro système est détecté.

Les paramètres passés à FETCH sont :

a) LIBRARY

Zone en mémoire où le corps de la macro système devra être placé.

b) NOM

Nom de la macro à chercher sur disque.

c) PTREC

Pointeur pour la première entrée libre dans LIBRARY, où

le premier bloc du corps de la macro sera placé. Cette valeur est extraite (voir la description de MACRANAL) de la pile DYNCORE.

d) RESULT

La valeur de RESULT précise le résultat de l'opération de lecture sur disque.

e) END1

C'est un indicateur, comme pour le cas de CREATE, qui provoquera la fermeture de la bibliothèque système la dernière fois que FETCH sera appelé.

De la même façon que CREATE, FETCH est écrit en assembleur et toutes les remarques au sujet du passage des paramètres entre le PL/I et l'assembleur, décrites dans l'item précédent, s'appliquent également pour le cas présent.

L'adresse sur disque de la macro appelée, est déterminée par la macro FIND (OS 360 macro) qui examine le répertoire et en extrait l'adresse du premier bloc à être lu.

Tous les autres blocs faisant partie du corps de la macro sont alors lus séquentiellement et placés dans les entrées correspondantes de LIBRARY. La lecture se termine lorsqu'on détecte l'instruction MEND qui détermine la fin du corps de la macro.

7. CHANGE1 et CHANGE

Le module CHANGE1 (écrit en PL/I) analyse le champ PARM, pour en extraire les anciens et les nouveaux noms de macros qui devront être changés dans le répertoire.

CHANGE1 active CHANGE (écrit en assembleur) en lui pas-

sant les deux noms comme paramètres. En utilisant la macro STOW, CHANGE effectue le changement dans le répertoire de la bibliothèque système.

8. DELETE1 et DELETE

Ces deux modules travaillent d'une façon entièrement analogue aux deux précédents. DELETE1 (écrit en PL/I) extrait de PARM les noms des macros à supprimer du répertoire et les passe à DELETE (écrit en assembleur), qui l'effectue à l'aide de la macro STOW.

9. Composition finale du système

Le système, comme nous l'avons déjà signalé, est constitué par 11 modules indépendants. Ces modules après avoir été compilés ou assemblés, sont introduits sur disque sous une forme non exécutable, car les références externes ne sont pas résolues après ce premier stade.

Cette procédure est assez commode, car on a toujours la possibilité d'effectuer des changements dans un ou plusieurs modules, sans être obligé de recompiler ou de réassembler tout le système.

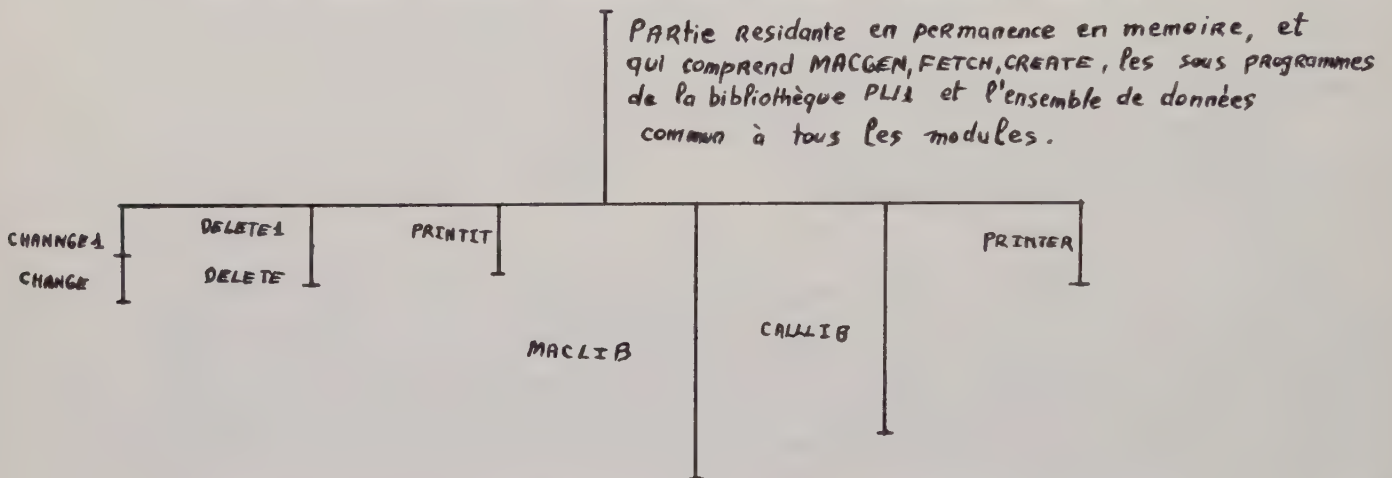
Pour constituer finalement le macro-générateur, il suffit de faire passer ces 11 modules par l'éditeur de liens qui résout toutes les références externes, en donnant comme résultat le système final sous forme exécutable laissé sur disque et catalogué sous le nom ZRAICKO.

Pendant l'exécution, les différents modules ne sont pas chargés en même temps en mémoire, ce qui demanderait au moins 250K octets pour tenir tout le système. Etant donné que plusieurs de ces modules travaillent indépendamment, une structure en "over-

lay" a été envisagée de manière à les charger en mémoire seulement au moment où ils sont appelés.

Les indications pour la disposition en mémoire des différents modules sont données à l'éditeur de liens au moment de les grouper ensemble.

La représentation schématique en arbre de l'espace occupé en mémoire par la structure en "overlay" est :



CHAPITRE III

CONCLUSIONS

- 1 - Lorsqu'on achève la mise au point d'un système, la première question qu'on doit se poser est :

La configuration finale qu'on a obtenue est-elle la plus efficace?

Pour bien répondre à cette question, il faut prendre en considération les contraintes principales qui ont déterminé les caractéristiques fondamentales du macro générateur et de son langage.

- a) Ce système doit être utilisé dans un centre équipé avec des machines IBM 360 et Telefunken TR 86. Ce sont des machines totalement différentes, non seulement dans leur architecture, mais surtout dans le niveau du "software" existant et les facilités offertes aux utilisateurs. D'une part les machines 360 avec son puissant système d'exploitation, parfois trop sophistiqué, et d'autre part les machines Telefunken pratiquement dépourvues jusqu'à présent d'un software de base suffisant pour les applications courantes. Etant donné que les usagers des deux machines peuvent se confondre, on a voulu produire un macro-générateur pour le TAS 86 aussi proche que possible dans son écriture du macro générateur de la 360, pour qu'on puisse passer facilement de l'un à l'autre. En gardant toutes les proportions, les deux macro générateurs sont assez semblables dans la façon dont ils manipulent les définitions de macros, les variables internes, les instructions d'assemblage conditionnel et les appels emboîtés.
- b) Le principal but du macro-générateur est d'être l'outil de base pour le développement d'un macro langage adapté aux prin-

cipaux besoins des systèmes temps - réel (faible facteur d'expansion, haute performance au point de vue temps d'exécution, etc). Cela veut dire que le maximum du travail d'analyse effectué par les macros-instructions faisant partie du macro langage, doit être fait pendant la génération. Pour pouvoir répondre à ces conditions, le macro générateur doit posséder des instructions qui permettent aux macros d'effectuer des analyses de caractères les plus variables, ainsi que des variables internes qui puissent stocker un grand nombre d'informations. Finalement, le langage du macro-générateur doit permettre une écriture très lisible de façon à rendre facile des changements ultérieurs éventuels dans les définitions de macros.

- c) Dans un projet comme le MADAP, les délais dans la production de programmes jouent un rôle fondamental. Le macro-langage, entièrement testé, doit être mis à la disposition des programmeurs le plus tôt possible, pour que les programmes d'application puissent démarrer.

Ces considérations ont déterminé la période de début mai 1970, pour la mise finale en opération du macro générateur et de sa bibliothèque système.

Les activités pour son développement ont commencé, en mai 1969, c'est-à-dire un an avant la date prévue pour sa mise en opération.

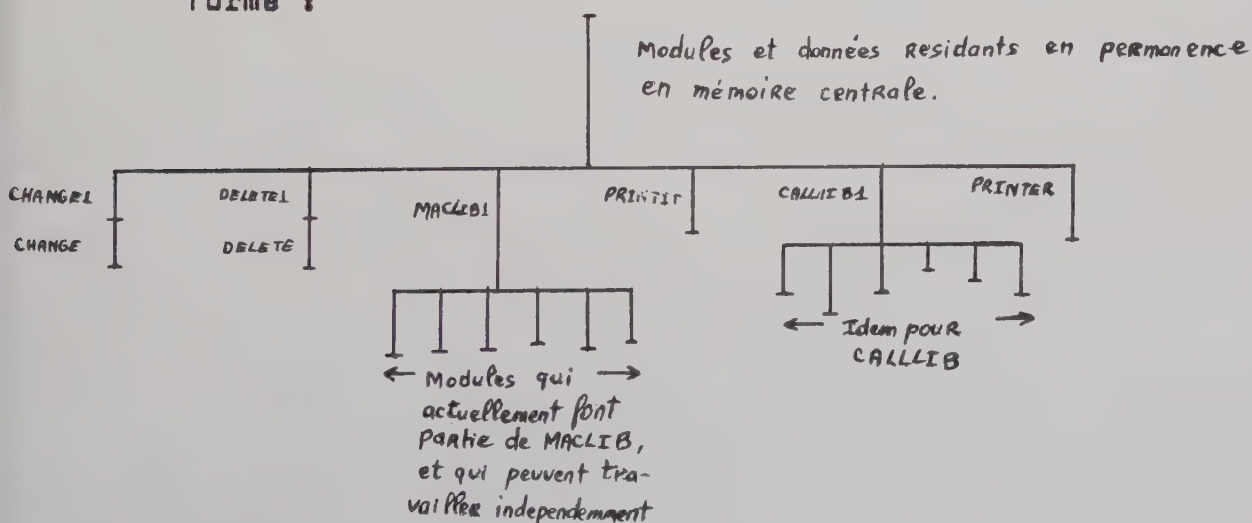
Le délai relativement court dont on disposait a été un des facteurs les plus importants pour déterminer une méthode d'action afin de mettre le système en fonctionnement avant la date prévue.

On a choisi le PL/I, car il s'adaptait très bien au type d'analyse à faire. Il est certain, que si tout le programme avait du être écrit en assembleur il ne serait jamais terminé en temps utile. D'autre part, on n'a pas cherché à optimiser le codage en profondeur, car on disposait de machines de 512 K octets de mé-

moire centrale. Etant donné que le macro générateur est un système qui tourne "off-line", une analyse plus profonde afin d'optimiser le codage n'a pas été faite car en principe on recherchait un système opérationnel en temps utile.

Toutes ces considérations sont importantes d'une part pour expliquer les deux principaux défauts de ce macro générateur : encombrement relativement important (160 K octets) en mémoire centrale, son manque de généralité par rapport à d'autres traducteurs comme le GPM¹. D'autre part, aucun système de programmation n'est optimal dans une première version. Toute une série de mesures à adopter pour améliorer ses performances ne sont entrevues qu'au moment de la fin. Cela est une règle très générale dans le domaine du "software" de base ce qui explique les nouvelles versions qui sortent périodiquement pour les compilateurs, systèmes d'exploitation etc.

Les plus grands modules du macro-générateur (MACLIB et CALLLIB) ont une structure logiquement modulaire, ce qui permettrait de remplacer plusieurs "GO TO" par des "CALL" à des procédures indépendantes et qui pourraient occuper le même espace en mémoire centrale. Dans ce cas nous aurions un arbre un peu plus complexe pour décrire la structure en "overlay" et qui aurait la forme :



Cette première mesure, permettra de réduire à 130 K octets l'espace occupé en mémoire centrale par le système.

Une meilleure organisation du dictionnaire d'erreurs et des messages, ainsi qu'une révision du codage, réduira ce chiffre à 110 K. D'autre part, étant donné qu'une grande partie du système a été écrite en PL/I, si de futures versions du compilateur, comme il est déjà envisagé, produisent un code plus optimisé (facteur d'expansion plus réduit), une autre amélioration en résultera directement.

Nous avons essayé dans les paragraphes précédents de décrire en lignes générales ce que pourrait être la deuxième version de ce macro-générateur.

La remarque finale, et peut-être la plus importante, est que, du point de vue pratique, la version actuelle est capable de répondre aux besoins réels des utilisateurs. Tout ce qu'on peut envisager pour l'avenir, n'est que du raffinement.

AI-1

ANNEXE 1

TEXTE DES PROGRAMMES

MACGENO

```

PROCEDURE(PARM) OPTIONS(MAIN)
DCL SORTIE FILE STREAM
DCL DIGITS CHAR(10) INIT('0123456789') EXTERNAL
DCL 1 DIRECTO(20) EXTERNAL,
    2 NAME CHAR(6) INIT((20)(6)' '),
    2 ADRESSE FIXED BINARY INIT((20)0),
    2 SWITCH FIXED BINARY INIT((20)0)
DCL 1 LIBRARY(500) EXTERNAL,
    2 LABEL CHAR(15) INIT((500)(15)' '),
    2 CODE CHAR(15) INIT((500)(15)' '),
    2 OPERAND CHAR(50) INIT((500)(50)' ')
DCL LIBRARY1(500) CHAR(80) DEFINED LIBRARY
DCL LETTER CHAR(26) INIT('ABCDEFGHIJKLMNOPQRSTUVWXYZ') EXTERNAL
DCL ARGUM(0015,10) CHAR(7) EXTERNAL
DCL ZONE1 CHAR(80) EXTERNAL
DCL ANAL1(80) CHAR(1) DEFINED ZONE1
DCL ZONE CHAR(864) EXTERNAL
DCL ANAL(864) CHAR(1) DEFINED ZONE
DCL PTLIB FIXED BINARY(31) INIT(1) EXTERNAL
DCL PTDIR FIXED BINARY INIT(1) EXTERNAL
DCL PTDIR1 FIXED BINARY EXTERNAL
DCL ERRIND FIXED BINARY INIT(0) EXTERNAL
DCL PTOUT FIXED BINARY(31) INIT(1) EXTERNAL
DCL MCCALL FIXED BINARY INIT(0) EXTERNAL
DCL CARD FIXED BINARY INIT(1) EXTERNAL
DCL CATOP CHAR(1) INIT('8') EXTERNAL
DCL OPERA CHAR(23) INIT('EQ*NE*GT*GE*LT*LE*NG*NL') EXTERNAL
DCL FIRSTCALL FIXED BINARY INIT(0)
DCL ACTMAC CHAR(210) INIT((30)' *') EXTERNAL
DCL DYNCLRE(20) FIXED BINARY EXTERNAL
DCL PTUYN FIXED BINARY INIT(1) EXTERNAL
DCL PTREC FIXED BINARY
DCL EXTRN FIXED BINARY EXTERNAL
DCL PTACT FIXED BINARY INIT(1) EXTERNAL
DCL PARM CHAR(50) VARYING
DCL CREIND FIXED BINARY(31) INIT(0)
DCL SOURIND FIXED BINARY INIT(0)
DCL END1 FIXED BINARY INIT(0)
DCL PAR2 CHAR(50)
DCL RESULT FIXED BINARY INIT(0)
DCL STRING1 CHAR(31) INIT('DCLA *DCLA *DCLC *DCLC ')
EXTERNAL
DCL IDEIND FIXED BINARY INIT(0)
DCL COMMENT CHAR(5) INIT(',-, ') EXTERNAL
DCL DUMPIND FIXED BINARY INIT(0) EXTERNAL
DCL NOPUNCH FIXED BINARY INIT(0) EXTERNAL
DCL LISTIND FIXED BINARY INIT(0)
DCL RECORD CHAR(7) EXTERNAL
DCL ANAL2(7) CHAR(1) DEFINED RECORD
DCL PTARG FIXED BINARY EXTERNAL
DCL (I,J,K,NCA) FIXED BINARY EXTERNAL
DCL LABIND FIXED BINARY EXTERNAL
DCL CSTART FIXED BINARY INIT(16) STATIC
DCL CL FIXED BINARY EXTERNAL
DCL CARDNUMBER FIXED BINARY INIT(1) STATIC
DCL NOM CHAR(6) EXTERNAL
DCL (IND,BLANK,POINT) FIXED BINARY EXTERNAL
DCL ATCARD FIXED BINARY EXTERNAL
DCL CARLENGTH FIXED BINARY INIT(68) STATIC
DCL SEP CHAR(1) EXTERNAL
DCL FIN FIXED BINARY INIT(0)
DCL CAT FIXED BINARY EXTERNAL
DCL INT FIXED BINARY EXTERNAL
DCL NOM1 CHAR(6) EXTERNAL
DCL NCB FIXED BINARY EXTERNAL
DCL PNI FIXED BINARY EXTERNAL
DCL TLENGTH FIXED BINARY INIT(68) STATIC
DCL CINIT FIXED BINARY
DCL NEST FIXED BINARY EXTERNAL
DCL ADDR FIXED BINARY EXTERNAL
DCL TYPE FIXED BINARY EXTERNAL
DCL 1 GLOCHAR(100) EXTERNAL,
    2 IDGCHAR CHAR(6) INIT((100)(6)' '),
    2 VAGCHAR CHAR(15) INIT((100)(15)' ')
DCL 1 GLOARIT(110) EXTERNAL,

```

```

      2 IDGDAR CHAR(6) INIT('1101(6)' ),
      2 VAGDAR FIXED BINARY(31) INIT('11010)
DCL PTGBAR FIXED BINARY INIT(1) EXTERNAL
DCL PTGBCH FIXED BINARY INIT(1) EXTERNAL
DCL WARN2 CHAR(13) INIT('NAME TOO LONG')EXTERNAL
DCL WARN3 CHAR(27) INIT('ONLY THE FIRST 6 WERE TAKEN')EXTERNAL

DCL WARN4 CHAR(19) INIT('IT HAS BEEN IGNORED') EXTERNAL
DCL WARN5 CHAR(20) INIT('IT HAS BEEN INSERTED') EXTERNAL
DCL PRINT1 CHAR(19) INIT('IN STATEMENT NUMBER') EXTERNAL
DCL PRINT2 CHAR(32) INIT('THE INSTRUCTION HAS BEEN DELETED')
EXTERNAL
DCL PRINT7 CHAR(8) INIT('OF MACRO') EXTERNAL
DCL ERROR1 CHAR(23) INIT('INVALID FIRST CHARACTER')EXTERNAL
DCL ERROR2 CHAR(15) INIT('STRING TOO LONG') EXTERNAL
DCL ERROR3 CHAR(44) INIT('LABELLED INSTRUCTION WITHOUT OPERATI
ON FIELD') EXTERNAL
DCL ERROR4 CHAR(27) INIT('INSTRUCTION WITHOUT OPERAND')
EXTERNAL
DCL ERROR5 CHAR(37) INIT('OPERATION CODE EXCEEDING 4 CHARACTERS
')
DCL ERROR6 CHAR(21) INIT('EQUAL SIGN MISSING IN')
DCL ERROR7 CHAR(27) INIT('INCORRECT USE OF EQUAL SIGN')
EXTERNAL
DCL ERROR9 CHAR(27) INIT('NAME NOT STARTING BY LETTER')
EXTERNAL
DCL ERROR11 CHAR(20) INIT('MACRO'S NAME MISSING')
DCL ERROR12 CHAR(26) INIT('MACRO'S NAME STARTING BY 1')
DCL ERROR13 CHAR(34) INIT('FORMAL PARAMETER NOT STARTING BY 1'
)
DCL ERROR131 CHAR(32) INIT('NUMBER TOO BIG OF SEQUENCE CARDS')
DCL ERROR14 CHAR(33) INIT('NUMBER OF PARAMETERS EXCEEDING 15')
DCL ERROR15 CHAR(13) INIT('ABNORMAL END') EXTERNAL
DCL ERROR16 CHAR(23) INIT('COMMA SEPARATOR MISSING')
EXTERNAL
DCL ERROR17 CHAR(17) INIT('MACRO NOT DEFINED')
DCL ERROR19 CHAR(28) INIT('STRING DEFINITION TOO LONG')
DCL ERROR20 CHAR(44) INIT('END OF CARD FOUND BEFORE STRING DEL
IMITATION')
DCL ERROR21 CHAR(17) INIT('INVALID SEPARATOR')
DCL ERROR22 CHAR(20) INIT('TOO MANY APOSTROPHES')
DCL ERROR23 CHAR(42) INIT('APOSTROPHES MATCHING COULD NOT BE A
CHIEVED')
DCL ERROR24 CHAR(21) INIT('INVALID CONCATENATION') EXTERNAL
DCL ERROR42 CHAR(20) INIT('HAS NOT BEEN DEFINED') EXTERNAL
ON ENDFILE(SYSIN) BEGIN
PUT SKIP EDIT('END OF SOURCE STATEMENT MISSING.IT HAS BEEN INSE
RTED')(A)
      RECORD='ND,
      GOTO A11 END
ON ERROR BEGIN
DCL UNIT CHAR(8)
UNIT='MACGEN'
PUT SKIP EDIT('AN ERROR OCURRED IN PROC',UNIT)
      (A,X(1),A)
PUT SKIP EDIT('INDICATIVE DATA--')(A)
PUT SKIP
PUT DATA(I,J,PTOUT)
NOPUNCH=2
IF DUMPIND=1 THEN DO
1 CALL IHEDUMJ
1 END
GOTO NEXTCARD
END
OPEN FILE(SORTIE) OUTPUT
IF INDEX(PARM,'DUMP') =0 THEN DUMPIND=1
IF INDEX(PARM,'CREATE') =0 THEN DO
1 CREIND=1
1 OPT20
LISTIND,NOPUNCH=1
GOTO OPT1
1 END
1 IF INDEX(PARM,'UPDATE') =0 THEN DO
1 CREIND=2

```

```

1      GOTO OPT2
1      END
      IF INDEX(PARM,'DELETE') =0 THEN DO
1      CREIND=3
1      OPT30SOURIND=1
1      GOTO OPT2
1      END
      IF INDEX(PARM,'CHANGE') =0 THEN DO
1      CREIND=4
1      GOTO OPT3
1      END
      IF INDEX(PARM,'NOSOURCE') =0 THEN
      SOURIND=1
      IF INDEX(PARM,'NODECK') =0 THEN
      NOPUNCH=1
      IF INDEX(PARM,'NOLIST') =0 THEN
      LISTIND=1
      IF INDEX(PARM,'IDENT') =0 THEN
      IDEIND=1
      OPT10
      CALL PRINTIT(SOURIND,CREIND,LISTIND,IDEIND)
      IF CREIND=3 THEN PAR2=PARM
      IF CREIND=3 THEN DO
1      CALL DELETE1(PAR2)
1      GO TO FIM
1      END
      ELSE
      IF CREIND=4 THEN DO
1      CALL CHANGE1(PAR2)
1      GO TO FIM
1      END
      /*INITIALISATIONS*/
      NEST=0
      CL=CARDLENGTH
      CINIT=CL
      PUT PAGE
NEXTCARD LABIND=0
      EXTRN=0
      CL=CARDLENGTH
      ZONE1=' '
      GET EDIT(ZONE) (A(80)) /*READING A CARD*/
      IF SOURIND=0 THEN
      PUT SKIP EDIT(CARD,ZONE)(F(4),X(3),A(80))
      ATCARD=CARD
      CARD=CARD+1 /*INCREASING THE CARD COUNTER*/
      /*INITIATE CHARACTER ANALYSIS*/
      I=1
      ALABEL10IF ANAL(I) =' ' THEN GO TO A1
      I=I+1
      IF I)=TLENGTH THEN GO TO ALABEL1
      GO TO NEXTCARD
      A10
      /*ANALYSING THE FIRST CHARACTER*/
      SEP=ANAL(I)
      IF INDEX(LETTER,SEP) =0 THEN GOTO A2
      IF INDEX(DIGITS,SEP) =0 THEN GOTO A19
      IF SEP='-' THEN GOTO A19
      IF SEP=''' THEN GOTO A19
      IF SEP='*' THEN DO
1      J=I
1      GO TO A25 END
      PUT SKIP
      PUT LIST(ERROR1,PRINT1,ATCARD,PRINT2)
      GO TO NEXTCARD
      A20 NCA=1
      ALABEL20
      IND=I+NCA
      SEP=ANAL(IND)
      IF SEP=' ' THEN GO TO A6
      IF SEP='*' THEN GO TO A7
      NCA=NCA+1
      IF IND)=TLENGTH THEN GO TO ALABEL2
      PUT SKIP
      PUT LIST(ERROR2,PRINT1,ATCARD,PRINT2)

```

SZ/

Z/

Z/1

Z/1

Z/1

Z/1

Z/1

Z/1

```

      GO TO NEXTCARD
A70LABIND=1
A60
      IF NCA)7 THEN GO TO A12
      IF NCA,7 THEN GO TO ALABEL5
      IF ANAL(I+6)='.' THEN GO TO ALABEL6
      /*FIELD EXTRACTION*/
      RECORD=SUBSTR(ZONE,I,7)
      GO TO A13
ALABEL50IF ANAL(I+NCA-1)='.' THEN GO TO ALABEL7
ALABEL60 /*FIELD EXTRACTION*/
      RECORD=SUBSTR(ZONE,I,6)
      GO TO A8
ALABEL70/*FIELD EXTRACTION*/
      RECORD=SUBSTR(ZONE,I,6)  '.'
      GO TO A8
A80  PUT SKIP
      PUT LIST(WARN2,PRINT1,ATCARD,WARN3)
      GO TO A13
A120RECORD=SUBSTR(ZONE,I,NCA) /*FIELD EXTRACTION*/
      IF NCA=3 THEN
      IF RECORD='ND, ' THEN GOTO A11
A130
      J=IND+1
ALABEL80 /*SEARCH FOR THE SECOND FIELD*/
      IF ANAL(J)=' ' THEN GO TO A16
      J=J+1
      IF J)TLENGTH THEN GOTO ALABEL4
      IF LABIND=1 THEN GO TO ALABEL9
A300
      IF RECORD='MACRO ' THEN DO
1      CALL MACRANAL
1      CALL MACLIB(MACRANAL,SOURIND)
1      IF CREIND=0 THEN GOTO NEXTCARD
1      ELSE GO TO AEX1
1      END
      GO TO A14
ALABEL90 PUT SKIP
      PUT LIST(ERROR3,PRINT1,ATCARD,PRINT2)
      GO TO NEXTCARD
A170/*INSERTION OF INSTRUCTION LABEL IN THE OUTPUT FILE*/
      SUBSTR(ZONE1,1,7)=RECORD
      ANAL(8)='='
      IF LABIND=1 THEN GOTO ALABEL14
ALABEL100J=J+1
      IF ANAL(J)=' ' THEN GO TO ALABEL11
      IF J)TLENGTH THEN GOTO ALABEL10
      PUT SKIP
      PUT LIST(ERROR3,PRINT1,ATCARD,PRINT2)
      GO TO NEXTCARD
ALABEL110
      IF INDEX(COMMENT,SUBSTR(ZONE,J,2))=0 THEN
      GOTO ALABEL9
      IF ANAL(J)='*' THEN GOTO A22
ALABEL140
      I=J GOTO A18
A140 PUT SKIP
      PUT LIST(ERROR4,PRINT1,ATCARD)
      ZONE1=SUBSTR(ZONE,1,TLENGTH)
      GO TO A20
A160
      SEP=ANAL(J)
      IF INDEX(COMMENT,SUBSTR(ZONE,J,2))=0 THEN
      IF LABIND=1 THEN GOTO ALABEL9
      ELSE GOTO A30
      IF SEP='=' THEN GOTO ALABEL12
      IF SEP='*' THEN GOTO ALABEL13
      IF LABIND=1 THEN GO TO A17
      IF NCA)=4 THEN DO
1      SUBSTR(ZONE1,I,NCA)=RECORD
1      I=J GOTO A19  END
      PUT SKIP
      PUT LIST(ERROR5,PRINT1,ATCARD,PRINT2)
      GO TO NEXTCARD

```

Z/

Z/

Z/1

Z/1

Z/

F/1

F/1

F/

F/

F/

F/1

F/1

F/1

F/1

F/1

```

ALABEL130 IF LABIND=1 THEN GO TO A25
          PUT SKIP
          PUT LIST(ERROR6,PRINT1,ATCARD,WARN5)
          GO TO A25
ALABEL120 IF LABIND =1 THEN GO TO A17
          PUT SKIP
          PUT LIST(ERROR7,PRINT1,ATCARD,PRINT2)
          GO TO NEXTCARD
A220
  LABIND=1
A250 MGCALL=1
CALL MACRANAL
  IF ERRIND=1 THEN ERRIND=0
  ELSE
CALL CALLLIB
  GO TO NEXTCARD
A190
  SUBSTR(ZONE1,I,TLENGTH+1-I)=SUBSTR(ZONE,I)
A200
  PUT FILE(SORTIE) EDIT(ZONE1) (A(80))
  PTOUT=PTOUT+1
  GO TO NEXTCARD
A180
  SUBSTR(ZONE1,10,TLENGTH-9)=SUBSTR(ZONE,I)
  GO TO A20
A110
  SUBSTR(ZONE1,10,4)=RECORD
  PUT FILE(SORTIE) EDIT(ZONE1) (A(80))
  END1=1
  IF CREIND =0 THEN
    CALL CREATE(LIBRARY1,NOM1,PTLIB,END1)
  ELSE
    CALL FETCH(LIBRARY1,NOM1,PTREC,RESULT,END1)
    CLOSE FILE(SORTIE)
    IF CREIND=0 THEN
      CALL PRINTER(SOURIND,CREIND,LISTIND,IDEIND)
      GOTO FIM
AEX10
  CALL CREATE(LIBRARY1,NOM1,PTLIB,END1,RESULT,CREIND)
  DO I=1 BY 1 TO PTLIB-1
    LIBRARY1(I)=' '
  END
  IF CREIND=1 THEN DO
    IF RESULT =0 THEN GOTO AEX5
AEX40
  PUT SKIP EDIT('THE MACRO',NOM1,',HAS BEEN INTRODUCED INTO THE L
IBRARY')(A,X(1),A,X(1),A)
AEX30
  PTLIB=1
  GO TO NEXTCARD
AEX50 IF RESULT=1 THEN DO
  PUT SKIP EDIT('I/O ERROR DETECTED DURING INSERTION OF MACRO',NO
M1,',IN THE LIBRARY')(A,X(1),A,X(1),A)
  GO TO AEX3      END
  IF RESULT=4 THEN DO
  PUT SKIP EDIT('THE MACRO',NOM1,',ALREADY EXISTED IN THE LIBRARY
')(A,X(1),A,X(1),A)
  GO TO AEX3      END
  IF RESULT=12 THEN DO
  PUT SKIP EDIT('NO SPACE IN THE DIRECTORY FOR MACRO',NOM1)(A,X(1
),A)
  GO TO AEX3      END
AEX5A0
  IF RESULT=16 THEN DO
  PUT SKIP EDIT('PERMANENT I/O ERROR,WHEN UPDATING THE DIRECTORY
FOR MACRO',NOM1) (A,X(1),A)
  GO TO AEX3      END
  PUT SKIP EDIT('MEANINGLESS RETURN CODE FOR MACRO',NOM1) (A,X(1
),A)
  GO TO AEX3      END
  IF RESULT=0 THEN DO
  PUT SKIP EDIT('THE MACRO',NOM1,',HAS BEEN REPLACED IN THE LIBRA
RY')(A,X(1),A)
  GO TO AEX3

```

F/1

F/1

F/1

F/1

F/1

F/1

```

1      END
      IF RESULT=1 THEN GOTO AEX5
      IF RESULT=8 THEN DO
1      PUT SKIP EDIT('THE MACRO',NOM1,', IS ABSENT IN THE LIBRARY.IT HA
S BEEN INSERTED') (A,X(1),A,X(1),A)
1      GO TO AEX3      END
      GO TO AEX5A
MACRANALOPROCEDURE /*IT ANALYSES AND EXTRACTS PARAMETERS IN A
                    MACRO DEFINITION OR MACRO CALL*/
      ON ERROR BEGIN
      DCL UNIT CHAR(8)
      UNIT='MACRANAL'
      PUT SKIP EDIT('AN ERROR OCURRED IN PROC',UNIT)
        (A,X(1),A)
      PUT SKIP EDIT('INDICATIVE DATAO')(A)
      PUT SKIP
      PUT DATA(PTARG,ATCARD,I,J)
      PUT SKIP EDIT('REQUIRED DUMPO')(A)
      CALL IHEDUMJ
      IF MCCALL=1 THEN GOTO B26
      ELSE GOTO ERRDEF
      END
/*INITIALISATIONS*/
      PTARG=0
      ARGUM(*,1)=' '
      FIN=0
      CARDNUMBER=1
      IF MCCALL=1 THEN GO TO BLAB1
B40LABIND=0
      GET EDIT(ZONE) (A(80))
      IF SOURIND=0 THEN
      PUT SKIP EDIT(CARD,ZONE) (F(4),X(3),A(80))
      ATCARD=CARD
      CARD=CARD+1
      BLAB10IF ANAL(72)=' ' THEN GO TO B3
      I=CINIT-CSTART+1
      /*READING CONTINUATION CARD*/
      B20 GET EDIT(ZONE1) (A(80))
      IF SOURIND=0 THEN
      PUT SKIP EDIT(ZONE1)(X(7),A(80))
      CARDNUMBER=CARDNUMBER+1
      /*ADDING INF TO ZONE*/
      SUBSTR(ZONE,CL+1,I)=SUBSTR(ZONE1,CSTART,I)
      CL=CL+I
      IF ANAL(72)=' ' THEN GO TO B3
      IF CARDNUMBER>15 THEN GO TO B2
      PUT SKIP
      PUT LIST(ERROR131,PRINT1,ATCARD,PRINT2)
      GOTO ERRDEF
      B30IF MCCALL=1 THEN GO TO B25
      I=1      /*SEARCH FOR THE FIRST CHARACTER*/
      BLAB20 IF ANAL(I)=' ' THEN GO TO B5
      I=I+1
      IF I=CL THEN GO TO BLAB2
      GO TO B4
      B50 IF ANAL(I)='£' THEN DO
1      J=I GO TO B11 END
      I=I+1 \
      IF INDEX(LETTER,ANAL(I))=0 THEN DO
1      PUT SKIP
1      PUT LIST(ERROR1,PRINT1,ATCARD,PRINT2)
1      GOTO ERRDEF
1      END
      NCA=1
      BLAB30IND=I+NCA
      SEP=ANAL(IND)
      IF SEP='.' THEN GO TO B7
      IF SEP='*' THEN GO TO B8
      NCA=NCA+1
      IF IND<CL THEN GO TO BLAB3
      PUT SKIP
      PUT LIST(ERROR2,PRINT1,ATCARD,PRINT2)
      GOTO ERRDEF
      B80 LABIND=1

```

```

      B120 IF NCA,6 THEN DO
1       PUT SKIP LIST(WARN2,PRINT1,ATCARD,WARN3)
1       NCA=6          END
      ARGUM(PTARG,1)=SUBSTR(ZONE,I,NCA)
      J=IND+1
BLAB40 IF ANAL(J) =' ' THEN GO TO B1
      J=J+1
      IF J)=CL THEN GO TO BLAB4
      B120 PUT SKIP
      PUT LIST(ERROR11,PRINT1,ATCARD,PRINT2)
      GOTO ERRDEF
      B90IF ANAL(J) ='=' THEN GO TO B10
      IF LABIND=1 THEN DO
1       PUT SKIP
1       PUT LIST(ERROR7,PRINT1,ATCARD,PRINT2)
1       GOTO ERRDEF
1       END
      J=J+1
BLAB50IF ANAL(J) =' ' THEN GO TO B11
      J=J+1
      IF J,CL THEN GO TO B12
      ELSE GO TO BLAB5
      B100 IF LABIND=1 THEN GO TO B11
      IF ANAL(J)='&' THEN DO
1       PUT SKIP
1       PUT LIST(ERROR12,PRINT1,ATCARD,PRINT2)
1       GOTO ERRDEF
1       END
      PUT SKIP
      PUT EDIT(ERROR6,PRINT1,ATCARD,WARN5)(A(19),A(19),F(4),A(20))
      B110IF INDEX(LETTER,ANAL(J))=0 THEN DO
1       PUT SKIP
1       PUT LIST(ERROR9,PRINT1,ATCARD,PRINT2)
1       GO TO NEXTCARD  END
      NCA=1
BLAB60IND=J+NCA
      IF ANAL(IND)=' ' THEN GO TO B13
      NCA=NCA+1
      IF IND)CL THEN GO TO BLAB6
      PUT SKIP
      PUT LIST(ERROR2,PRINT1,ATCARD,PRINT2)
      GOTO ERRDEF
      B130IF NCA,6 THEN DO
1       PUT SKIP
1       PUT LIST(WARN2,PRINT1,ATCARD,WARN3)
1       NCA=6          END
      NOM1=SUBSTR(ZONE,J,NCA)
      IF CREIND=0 THEN DO
1       NAME(PTDIR)=NOM1
1       ADRESSE(PTDIR)=PTLIB
1       PTDIR=PTDIR+1
1       END
      J=IND+1
      BLAB70IF ANAL(J) =' ' THEN GO TO B15
      J=J+1
      IF J)=CL THEN GO TO BLAB7
      B140
      ADDR=PTLIB-1
      RETURN
      B150 IF ANAL(J) ='&' THEN DO
1       B15A0
      PUT SKIP
1       PUT LIST(ERROR13,PRINT1,ATCARD,PRINT2)
1       GOTO ERRDEF
1       END
      B160J=J+1
      IF INDEX(LETTER,ANAL(J))=0 THEN DO
1       PUT SKIP
1       PUT LIST(ERROR9,PRINT1,ATCARD,PRINT2)
1       GO TO NEXTCARD  END
      NCA=1
BLAB80
      IND=J+NCA
      SEP=ANAL(IND)

```

```

      IF IND+1=CL THEN GOTO BEX10
      IF SEP=' ' THEN GOTO B17
      IF SEP=', ' THEN GOTO B18
      NCA=NCA+1
      GOTO BLAB8
BEX100
      IF SEP=', ' THEN GOTO BEX12
      IF ANAL(CL)=' ' THEN GOTO B22
BEX110
      FIN=1
      GOTO B20
BEX120
      IF SEP=', ' THEN GOTO B22
      SEP=ANAL(CL)
      IF SEP=', ' THEN DO
1      NCA=NCA+1 GO TO BEX11 END
      IF SEP=', ' THEN GO TO B22
      NCA=NCA+2
      GO TO BEX11
B170 BLANK=1
      GO TO B20
B180 BLANK=0
B200 IF NCA,6 THEN DO
1      PUT SKIP
1      PUT LIST(WARN2,PRINT1,ATCARD,WARN3)
1      NCA=6 END
      PTARG=PTARG+1
      IF PTARG, 15 THEN DO
1      BEX10PUT SKIP
1      PUT LIST(ERROR14,PRINT1,ATCARD,PRINT2)
1      IF MCCALL=1 THEN GOTO B26
1      GOTO ERRDEF
1      END
      ARGUM(PTARG,1)=SUBSTR(ZONE,J,NCA)
      IF FIN=1 THEN DO
1      FIN=0 GO TO B14 END
      J=IND+1
      IF BLANK =0 THEN GO TO B19
B1220 IF ANAL(J)=' ' THEN GO TO B15
      J=J+1
      IF J)=CL THEN GO TO B122
B220 PUT SKIP
      PUT LIST(ERROR15,PRINT1,ATCARD,PRINT2)
      IF MCCALL=1 THEN GOTO B26
      GOTO ERRDEF
B190
      SEP=ANAL(J)
      IF SEP=' ' THEN GOTO B21
      J=J+1
      IF J)=CL THEN GO TO B19
      GO TO B14
B210
      IF SEP=', ' THEN DO
1      J=J+1
1      GO TO B122 END
      PUT SKIP
BEX130 PUT LIST(ERROR16,PRINT1,ATCARD,PRINT2)
      IF MCCALL=1 THEN GOTO B26
      ERRDEF0
      NOPUNCH=2
      GO TO NEXTCARD
B250
      IF LABIND=1 THEN DO
1      IF NEST=0 THEN
1      ARGUM(PTARG,1)=RECORD
1      ELSE IF CAT=1 THEN DO
2      PUT SKIP LIST(ERROR24,PRINT1,ATCARD,PRINT2)
2      GOTO B26
2      END
      END
      J=J+1
      IF INDEX(LETTER,ANAL(J))=0 THEN GO TO BLAB10
      NCA=1
BLAB90 IND=J+NCA
      IF ANAL(IND)=' ' THEN GO TO B27

```

```

      NCA=NCA+1
      IF IND)CL THEN GO TO BLAB9
      PUT SKIP
      PUT LIST(ERROR2,PRINT1,ATCARD,PRINT2)
      GO TO B26
BLAB100PUT SKIP
      PUT LIST(ERROR9,PRINT1,ATCARD,PRINT2)
      B260 MDCALL=0
      NOPUNCH=2
      ERRIND=1
      RETURN
      B270 IF NCA,6 THEN DO
1         PUT SKIP
1         PUT LIST(WARN2,PRINT1,ATCARD,WARN3)
1         NCA=6          END
          NOM=SUBSTR(ZONE,J,NCA)
      IF NEST=1 THEN DO
1         CODE(PTLIB)=*' ' NOM
1         GOTO BBLAB12A    END
          /*SEARCH FOR THE MACRO*/
          I=1
BLAB120  IF NAME(I)=NOM THEN GO TO BLAB11
          I=I+1
          IF I)PTDIR THEN GO TO BLAB12
          GOTO B61
          IF FIRSTCALL=0 THEN GOTO B60
BLAB110PTLIB=ADRESSE(I)
          PTDIR1=I
BLAB120  ADDR=PTLIB-1
          BBLAB12A0
          J=IND+1
BLAB1210 IF ANAL(J) =' ' THEN GO TO B28
          J=J+1
          IF J)=CL THEN GO TO BLAB121
      B290
          MDCALL=0
          IF NEST =0 THEN RETURN
          PRINTCALL0
          I,J=1
          CL=CARDLENGTH
          PRCALL10
          ZONE1='--' SUBSTR(ZONE,I,CL)
          PUT FILE(SORTIE) EDIT(ZONE1) (A(80))
          PTOUT=PTOUT+1
          IF J)CARDNUMBER THEN DO
1             I=I+CL
1             J=J+1
1             GOTO PRCALL1    END
          RETURN
      B280
          SEP=ANAL(J)
          IF SEP=''' THEN GOTO B30
          IF SEP=', ' THEN GOTO B40
          NCA=1
BLAB130
          IND=J+NCA
          SEP=ANAL(IND)
          IF IND+1=CL THEN GOTO BEX2
          IF SEP=' ' THEN GOTO B31
          IF SEP=', ' THEN GOTO B32
          NCA=NCA+1
          GOTO BLAB13
      BEX20
          IF SEP=', ' THEN GOTO BEX5
          IF SEP=' ' THEN
          IF ANAL(CL)=' ' THEN GOTO BEX3
          ELSE GOTO B22
          SEP=ANAL(CL)
          IF SEP=' ' THEN GOTO BEX3
          IF SEP=', ' THEN GOTO BEX4
          GOTO B22
      BEX40NCA=NCA+1
      BEX30

```

SZ/1

SZ/1

SZ/1

SZ/1

SZ/1

SZ/1

SZ/1

SZ/1

SZ/1

```

      FIN=1
      GOTO B33
BEX50  IF ANAL(CL)=' ' THEN GOTO B22
      FIN=2
      GOTO B33
B310   BLANK=1
      GOTO B33
B320   BLANK=0
      B330PTARG=PTARG+1
      IF PTARG,15 THEN GOTO BEX1
      ARGUM(PTARG,1)=SUBSTR(ZONE,J,NCA)
B340   IF FIN =0 THEN DO
1       IF FIN=2 THEN DO
2       PTARG=PTARG+1
2       IF PTARG,15 THEN GOTO BEX1
2       ARGUM(PTARG,1)=SUBSTR(ZONE,CL,1)
2       FIN=0
2       GOTO B29
1       END
1       FIN=0      GOTO B29
1       END
J=IND + 1
      IF BLANK =0 THEN GO TO B37
B390IF ANAL(J) =' ' THEN GO TO B28
      J=J+1
      IF J)=CL THEN GO TO B39
      PUT LIST(ERROR15,PRINT1,ATCARD,PRINT2)
      GOTO B26
      B370
      SEP=ANAL(J)
      IF SEP =' ' THEN GOTO B38
      J=J+1
B420IF J)=CL THEN GO TO B37
      GO TO B29
B380   IF SEP=',' THEN DO
1       J=J+1  GO TO B39  END
      GO TO BEX13
B400 PTARG=PTARG+1
      IF PTARG,15 THEN GOTO BEX1
      J=J+1
      GO TO B39
B300RECORD=' '
      NCA=1
BLAB160 J=J+1
      IF ANAL(J)=''' THEN GO TO B41
B440IF J=CL THEN GO TO B43
      IF NCA,7 THEN DO
1       PUT SKIP
1       PUT LIST(ERROR19,PRINT1,ATCARD,PRINT2)
1       GO TO B26  END
      ANAL2(NCA)=ANAL(J)
      NCA=NCA+1
      GO TO BLAB16
B430PUT SKIP
      PUT LIST(ERROR20,PRINT1,ATCARD,PRINT2)
      GO TO B26
B410IF J=CL THEN GO TO B46
      IF ANAL(J+1)=''' THEN GO TO BLAB17
      ELSE GO TO B47
B460FIN=1
B470PTARG=PTARG+1
      IF PTARG,15 THEN GOTO BEX1
      ARGUM(PTARG,1)=RECORD
      IF FIN =0 THEN DO
1       FIN=0 GO TO B29  END
      J=J+1
      SEP=ANAL(J)
      IF SEP=' ' THEN GOTO B42
      IF SEP=',' THEN DO
1       J=J+1  GO TO B39  END

```

```

      PUT SKIP
      PUT LIST(ERROR21,PRINT1,ATCARD,PRINT2)
      GO TO B26
BLAB170 IF J+1=CL THEN GO TO B43
      IF ANAL(J+2)=''' THEN GO TO B45
      ANAL2(NCA)='''
      NCA=NCA+1
      J=J+2
      GO TO B44
      B450 IF J+2 =CL THEN GO TO BLAB18
      ANAL2(NCA)='''
      GO TO B46
BLAB180
      SEP=ANAL(J+3)
      IF SEP=' ' THEN GOTO B48
      IF SEP=', ' THEN GOTO B48
      ELSE GO TO B49
      B480 ANAL2(NCA)='''
      J=J+3
      GO TO B47
      B490
      IF SEP ='''' THEN GOTO B50
      IF J+3= CL THEN DO
1      FIN=1 GO TO B50 END
      SEP=ANAL(J+4)
      IF SEP=' ' THEN DO
1      BLAB190 FIN=2 GO TO B50 END
      IF SEP=', ' THEN GOTO BLAB19
      PUT SKIP
      PUT LIST(ERROR22,PRINT1,ATCARD,PRINT2)
      GO TO B26
      B500 ANAL2(NCA)='''
      NCA=NCA+1
      IF NCA,7 THEN GO TO BLAB20
      ANAL2(NCA)='''
      J=J+3
      IF NCA,7 THEN GO TO BLAB20
      IF FIN=0 THEN DO
1      NCA=NCA+1
1      GO TO B44 END
      IF FIN=1 THEN GO TO B47
      FIN=0
      GO TO B47
      BLAB200 PUT SKIP
      PUT LIST(ERROR23,PRINT1,ATCARD,PRINT2)
      GO TO B26
      B600
      FIRSTCALL=1
      DYNCORE(PTDYN)=PTLIB
      PTDYN=PTDYN+1
      GO TO BLAB11
      B610
      IF FIRSTCALL=0 THEN DO
1      DYNCORE(PTDYN)=PTLIB
1      PTDYN=PTDYN+1
1      FIRSTCALL=1 END
      PTREC=DYNCORE(PTDYN-1)
      CALL FETCH(LIBRARY1,NOM,PTREC,RESULT,END1)
      IF RESULT=0 THEN DO
1      DYNCORE(PTDYN)=PTREC
1      PTEIB=DYNCORE(PTDYN-1)
1      PTDYN=PTDYN+1
1      EXTRN=1
1      GO TO BBLAB12 END
      IF RESULT=4 THEN GOTO BLAB12A
      PUT SKIP EDIT('I/O ERROR DETECTED WHEN RETRIEVING MACRO',NOM)
      (A,X(1),A)
      GOTO B26
      BLAB12A0
      PUT SKIP
      PUT LIST(ERROR17,PRINT1,ATCARD,PRINT2)
      GO TO B26
      END
FIMO END

```

```

MACLIBO
PROCEDURE(MACANAL,SOURPARM)
DCL SOURPARM FIXED BINARY
DCL MACANAL ENTRY
DCL DIGITS CHAR(10) INIT('0123456789') EXTERNAL
DCL 1 LIBRARY(500) EXTERNAL,
    2 LABEL CHAR(15) INIT((500)(15)' '),
    2 CODE CHAR(15) INIT((500)(15)' '),
    2 OPERAND CHAR(50) INIT((500)(50)' ')
DCL LIBRARY1(500) CHAR(80) DEFINED LIBRARY
DCL 1 LABELTAB(100),
    2 LABNAME CHAR(6) INIT((100)(6)' '),
    2 LABADDR FIXED BINARY INIT((100)0)
DCL ARGUM(0015,10) CHAR(7) EXTERNAL
DCL LETTER CHAR(26) INIT('ABCDEFGHIJKLMNOPQRSTUVWXYZ') EXTERNAL
DCL ZONE CHAR(864) EXTERNAL
DCL ANAL(864) CHAR(1) DEFINED ZONE
DCL RECORD CHAR(7) EXTERNAL
DCL ANAL2(7) CHAR(1) DEFINED RECORD
DCL PTLIB FIXED BINARY(31) INIT(1) EXTERNAL
DCL PTARG FIXED BINARY EXTERNAL
DCL (I,J,K,NCA) FIXED BINARY EXTERNAL
DCL LABIND FIXED BINARY EXTERNAL
DCL CL FIXED BINARY EXTERNAL
DCL MCCALL FIXED BINARY INIT(0) EXTERNAL
DCL NOM CHAR(6) EXTERNAL
DCL (IND,BLANK,POINT) FIXED BINARY EXTERNAL
DCL CARD FIXED BINARY INIT(1) EXTERNAL
DCL CATOP CHAR(1) INIT('&') EXTERNAL
DCL ATCARD FIXED BINARY EXTERNAL
DCL SEP CHAR(1) EXTERNAL
DCL CAT FIXED BINARY EXTERNAL
DCL INT FIXED BINARY EXTERNAL
DCL BRANCH FIXED BINARY STATIC
DCL PTLAB FIXED BINARY STATIC
DCL NOM1 CHAR(6) EXTERNAL
DCL HEXA CHAR(16) INIT('0123456789ABCDEF') STATIC
DCL AUXOP2 CHAR(15) EXTERNAL
DCL AUXOP1 CHAR(7) EXTERNAL
DCL NCB FIXED BINARY EXTERNAL
DCL FORMAL FIXED BINARY STATIC
DCL AUXOP3 CHAR(50) EXTERNAL
DCL ANAL3(50) CHAR(1) DEFINED AUXOP3
DCL WORK CHAR(2) STATIC
DCL APOST FIXED BINARY STATIC
DCL SEPI CHAR(1) STATIC
DCL DUMPIND FIXED BINARY INIT(0) EXTERNAL
DCL CARDLENGTH FIXED BINARY INIT(68)
DCL ERRIND FIXED BINARY INIT(0) EXTERNAL
DCL PNT FIXED BINARY EXTERNAL
DCL ORDRE CHAR(2) STATIC
DCL COMMENT CHAR(5) INIT(',-, ') EXTERNAL
DCL STRING1 CHAR(31) INIT('DCLA *DCGA *DCLC *DCGC ')
EXTERNAL
DCL NEST FIXED BINARY EXTERNAL
DCL ADDR FIXED BINARY EXTERNAL
DCL TYPE FIXED BINARY EXTERNAL
DCL PARAM FIXED BINARY
DCL PTDEF FIXED BINARY
DCL OPERA CHAR(23) INIT('EQ*NE*GT*GE*LT*LE*NG*NL') EXTERNAL
DCL ARGUM1(0015) CHAR(7) /*WORKING ZONE*/
DCL (PTARGT,PTARG1) FIXED BINARY /*AUXILIARY POINTERS*/
DCL NOPUNCH FIXED BINARY INIT(0) EXTERNAL
DCL NUMERICS(0015) CHAR(2) INIT('00','01','02','03','04','05','06',
'07','08','09','10','11','12','13','14','15')
DCL 1 BRANCHCONT(100),
    2 BRDEF FIXED BINARY,
    2 INSPOINT FIXED BINARY
DCL STRING2 CHAR(15) INIT('CAB *UAB ')
DCL STRING3 CHAR(15) INIT('SETA *SETC ')
DCL WARN2 CHAR(13) INIT('NAME TOO LONG') EXTERNAL
DCL WARN3 CHAR(27) INIT('ONLY THE FIRST 6 WERE TAKEN') EXTERNAL
DCL WARN4 CHAR(19) INIT('IT HAS BEEN IGNORED') EXTERNAL
DCL WARN5 CHAR(20) INIT('IT HAS BEEN INSERTED') EXTERNAL
DCL PRINTI CHAR(19) INIT('IN STATEMENT NUMBER') EXTERNAL

```

```

      DCL PRINT2 CHAR(32) INIT('THE INSTRUCTION HAS BEEN DELETED')
EXTERNAL
      DCL PRINT3 CHAR(18) INIT('IN OPERATION FIELD')
      DCL PRINT4 CHAR(16) INIT('IN OPERAND FIELD')
      DCL PRINT5 CHAR(14) INIT('INTERNAL LABEL')
      DCL PRINT6 CHAR(20) INIT('NO BRANCH IS ASSUMED')
      DCL PRINT7 CHAR(8) INIT('OF MACRO') EXTERNAL
      DCL PRINT20A CHAR(11) INIT('REFERRED TO')
      DCL ERROR1 CHAR(23) INIT('INVALID FIRST CHARACTER') EXTERNAL
      DCL ERROR2 CHAR(15) INIT('STRING TOO LONG') EXTERNAL
      DCL ERROR3 CHAR(44) INIT('LABELLED INSTRUCTION WITHOUT OPERATI
ON FIELD') EXTERNAL
      DCL ERROR4 CHAR(27) INIT('INSTRUCTION WITHOUT OPERAND')
EXTERNAL
      DCL ERROR7 CHAR(27) INIT('INCORRECT USE OF EQUAL SIGN')
EXTERNAL
      DCL ERROR9 CHAR(27) INIT('NAME NOT STARTING BY LETTER')
EXTERNAL
      DCL ERROR15 CHAR(13) INIT('ABNORMAL END') EXTERNAL
      DCL ERROR16 CHAR(23) INIT('COMMA SEPARATOR MISSING')
EXTERNAL
      DCL ERROR24 CHAR(21) INIT('INVALID CONCATENATION') EXTERNAL
      DCL ERROR25 CHAR(37) INIT('INVALID CHARACTER IN NUMERIC CONSTAN
T')
      DCL ERROR26 CHAR(34) INIT('INVALID CHARACTER IN OPERAND FIELD')
      DCL ERROR27 CHAR(24) INIT('INVALID NUMERIC CONSTANT')
      DCL ERROR28 CHAR(28) INIT('INVALID INTERNAL DECLARATION')
      DCL ERROR29 CHAR(21) INIT('INVALID SET OPERATION')
      DCL ERROR30 CHAR(23) INIT('INVALID STRING CONSTANT')
      DCL ERROR31 CHAR(31) INIT('POINT MISSING IN INTERNAL LABEL')
      DCL ERROR32 CHAR(22) INIT('INTERNAL LABEL MISSING')
      DCL ERROR33 CHAR(31) INIT('INVALID CAB INSTRUCTION')
      DCL ERROR34 CHAR(35) INIT('INVALID OPERATOR IN CAB INSTRUCTION')
      DCL ERROR35 CHAR(21) INIT('INVALID SECOND OPERAND')
      DCL ERROR36 CHAR(21) INIT('MORE THAN 7 VARIABLES')
      DCL ERROR37 CHAR(31) INIT('INVALID INTERNAL REPRESENTATION')
      DCL ERROR38 CHAR(27) INIT('UNDEFINED FORMAL PARAMETERS')
      DCL ERROR39 CHAR(32) INIT('INVALID OPERAND IN MOVE FUNCTION')
      DCL ERROR40 CHAR(57) INIT('NUMBER OF INTERNAL LABEL REFERENCES
EXCEEDING 100 IN MACRO')
      DCL ERROR40A CHAR(49) INIT('INTERNAL LABEL DEFINITIONS EXCEEDIN
G 100 IN MACRO')
      DCL ERROR41 CHAR(26) INIT('UNDEFINED FORMAL PARAMETER')
      DCL ERROR42 CHAR(20) INIT('HAS NOT BEEN DEFINED') EXTERNAL
      DCL ERROR43 CHAR(20) INIT('DOUBLE DEFINITION OF')
      ON ENDFILE(SYSIN) BEGIN
      PUT SKIP EDIT('END OF FILE FOUND DURING THE ANALYSIS OF MACRO--',NOM1
) (A,A)
      PUT SKIP EDIT('CHECK FOR THE MEND STATEMENT')(A)
      GOTO RETURPT
      END
      ON ERROR BEGIN
      DCL UNIT CHAR(8)
      UNIT='MACLIB'
      PUT SKIP EDIT('AN ERROR OCCURRED IN PROC',UNIT)
      (A,X(1),A)
      PUT SKIP EDIT('INDICATIVE DATA--')(A)
      PUT SKIP EDIT('MACRO BEING ANALYSED--',NOM1)(A,A)
      PUT SKIP
      PUT DATATPTLIB,ATCARD,I,J,AUXOP3)
      NOPUNCH=2
      IF DUMPIND=1 THEN DO
      PUT SKIP EDIT('REQUIRED DUMP--')(A)
      CALL IHEDUMJ
      END
      GOTO C1
      END
/*INITIALISATIONS*/
      NEST=0
      CL=CARDLENGTH
      PTLAB=1 PTDEF=1
      C10 LABIND=0
      AUXOP2=' '
      AUXOP3=' '

```

1
1
1

```

NCB=0
CAT=0
INT=0
APOST=0
FORMAL=0
BRANCH=0
WORK=' '
/*READ A CARD*/
GET EDIT(ZONE) (A(80))
IF SOURPARM=0 THEN
  PUT SKIP EDIT(CARD,ZONE)(F(4),X(3),A(80)) /*PRINTING A CARD*/
  ATCARD=CARD
  CARD=CARD+1
  /*INITIATE CHARACTER ANALYSIS*/
  I=1
CLAB10 IF ANAL(I) = ' ' THEN GO TO C2
  I=I+1
  IF I)=CL THEN GO TO CLAB1
  GOTO C1
C20
  SEP=ANAL(I)
IF INDEX(LETTER,SEP) =0 THEN GOTO C3
  IF SEP='2' THEN GO TO C4
  IF SEP='''' THEN GO TO C6
  IF INDEX(DIGITS,SEP) =0 THEN GO TO C7
  IF SEP='-' THEN GO TO C120
  IF SEP='.' THEN DO
1      I=I+1
1      IF INDEX(LETTER,ANAL(I))=0 THEN GO TO CERR3
1      BRANCH=1
1      GO TO C3  END
  IF SEP='*' THEN GO TO NESTING
  IF SEP='+' THEN GOTO C7
  GO TO CERR2
C30 NCA=1
CLAB20
IND=I+NCA  SEP =ANAL(IND)
  IF SEP=' ' THEN GO TO C10
  IF SEP='=' THEN GO TO C11
  IF SEP=CATOP THEN GO TO C12
  NCA=NCA+1
  IF IND)CL THEN GO TO CLAB2
  GO TO CERR4
C120CAT=1
  IF BRANCH=1 THEN GO TO CERR5
  ELSE GO TO C10
C110LABIND=1
C100IF NCA)7 THEN GO TO C14
  IF BRANCH=1 THEN GO TO C13
  IF NCA=7 THEN DO
1      IF ANAL(I+6)='.' THEN GO TO C14
1      ELSE GO TO C13  END
  IF ANAL(I+NCA-1)='.' THEN POINT=1
C130NCA=6
  PUT SKIP
  PUT LIST(WARN2,PRINT1,ATCARD,WARN3)
C140RECORD=SUBSTR(ZONE,I,NCA)
  IF POINT=1 THEN DO
1      POINT=0
1      ANAL2(7)='.'
1      END
  IF BRANCH=1 THEN GO TO C9
  IF CAT=1 THEN GO TO C24
C150OI=IND+1
CLAB30IF ANAL(I) = ' ' THEN GO TO C17
  I=I+1
  IF I)=CL THEN GO TO CLAB3  IF LABIND =0 THEN GO TO CERR6
  CODE(PTLIB)=RECORD
  IF RECORD =*MEND  ' THEN GOTO C121
C230
  PTLIB=PTLIB+1
  IF PTDEF,1 THEN
    GO TO INBRANCH /*PROC FOR INSERTING ADDRESSES FOR CONDITIONN.
                     ASSEMBLER BRANCHES*/

```

```

RETURPT ORETURN
C90
  PNT=1
  CLAMT60
    IF LABNAME(PNT)=RECORD THEN DO
1      BRANCH=0
1      GOTO CERR25  END
    IF PNT)PTLAB THEN DO
1      PNT=PNT+1
1      GOTO CLAMT6  END
    IF PTLAB,100 THEN DO
1      PUT SKIP EDIT(ERROR40A,NOM1,PRINT1,ATCARD,PRINT2)
1      (A,X(1),A,X(1),A,F(4),X(1),A)
1      BRANCH=0
1      GO TO CERREX  END
    LABNAME(PTLAB)=RECORD
    LABADDR(PTLAB)=PTLIB
    PTLAB=PTLAB+1
    GO TO C15A
  C170
    IF ANAL(I)='*' THEN DO
1      I=I+1
1      GO TO CLAB4  END
    IF LABIND=1 THEN GO TO CLAB5
    IF CAT=1 THEN DO
1      NCB=1  GO TO C17A  END
    IF BRANCH=1 THEN GOTO CERR8
    CODE(PTLIB)=RECORD
    PNT=INDEX(STRING2,RECORD)
    IF PNT=1 THEN GO TO CONDBRANCH
    IF PNT=9 THEN GO TO UNCOBRANCH IF INDEX(STRING3,RECORD) =0 THEN
GO TO CERR14 NCB=INDEX(STRING1,RECORD)
    IF NCB =0 THEN GO TO INTDECL IF RECORD='I ' THEN GO TO IN
    IF RECORD='NOTE ' THEN GO TO CNOTE
    GO TO C18
  CLAB40IF LABIND=1 THEN GO TO CERR7 LABIND=1
  CLAB50IF CAT=1 THEN GO TO C17A
  C220
    IF BRANCH=0 THEN
      LABEL(PTLIB)=RECORD
  CLAB60 IF ANAL(I) = ' ' THEN GO TO CLAB7
    I=I+1
    IF I)=CL THEN GO TO CLAB6
    GO TO CERR6
  CLAB70 CAT=0 SEP=ANAL(I)
    IF SEP='Σ' THEN GO TO C60
    IF SEP='*' THEN GO TO NESTING
    IF INDEX(DIGITS,SEP) =0 THEN GO TO C7
    IF INDEX(LETTER,SEP) =0 THEN GO TO C50
    IF SEP='''' THEN GO TO C6
    IF SEP='-' THEN GO TO C120
    IF SEP='(' THEN GOTO C150
    IF SEP='+' THEN GOTO C7
    GO TO CERR8
  C17A0 IND=INDEX(AUXOP1,' ')
    IF IND=0 THEN IND=8
    AUXOP2=SUBSTR(AUXOP1,1,IND-1) CATOP RECORD
    GOTO C22B
  C40I=I+1
    IF ANAL(I)='Σ' THEN DO
1      INT=1 I=I+1  END
    IF INDEX(LETTER,ANAL(I))=0 THEN GO TO CERR3
    NCA=1
  CLAB80IND=I+NCA
    SEP=ANAL(IND)
    IF SEP=''' THEN GO TO C19
    IF SEP=' ' THEN GO TO C20
    IF SEP=CATOP THEN GO TO C21
    IF IND=CL THEN GO TO CERR4
    NCA=NCA+1
    GO TO CLAB8
  C22B0 IF NCB=0 THEN
    LABEL(PTLIB)=AUXOP2
    ELSE DO

```

```

1      CODE(PTLIB)=AUXOP2
1      GOTO C18      END
      GO TO CLAB6
C200   LABIND=1
      GO TO C19
C210   CAT=1
C190   IF NCA,6 THEN DO
1       PUT SKIP
1       PUT LIST(WARN2,PRINT1,ATCARD,WARN3)
1       NCA=6
1       END
      RECORD=SUBSTR(ZONE,I,NCA)
      IF INT =0 THEN DO
1       SUBSTR(RECORD,2)=RECORD
1       ANAL2(1)='2'      END
      ELSE CALL SEARARG
      IF CAT=1 THEN GO TO C40
      I=IND+1
CLAB90IF ANAL(I)=' ' THEN GO TO CLAB10
      I=I+1
      IF I)=CL THEN GO TO CLAB9
      IF LABIND =0 THEN GOTO CERR6
      PUT SKIP
      PUT LIST(ERROR4,PRINT1,ATCARD)
      CODE(PTLIB)=RECORD
      GOTO C121
CLAB100IF ANAL(I)='=' THEN DO
1       IF LABIND=1 THEN GO TO CERR7
1       I=I+1
1       LABIND=1      END
CLAMT10
      IF ANAL(I)=' ' THEN
      IF I=CL THEN GOTO CERR6
      ELSE DO
1       I=I+1
1       GOTO CLAMT1      END
      ELSE
      IF LABIND=1 THEN DO
1       LABEL(PTLIB)=RECORD
1       GO TO CLAB7      END
      ELSE DO
1       CODE(PTLIB)=RECORD
1       GO TO C18      END
C70   NCA=1
CLAB110IND=I+NCA
      SEP=ANAL(IND)
      IF INDEX(HEXA,SEP) =0 THEN GOTO CLAMT2
      IF SEP=' ' THEN
      IF APOST = 1 THEN GO TO CERR9
      ELSE GO TO C70
      IF SEP=CATOP THEN GO TO C71
      IF SEP='''' THEN DO
1       SEP=ANAL(IND+1)
1       IF SEP='H' THEN GO TO CLAB12
1       ELSE IF SEP='O' THEN GO TO CLAB12
1       ELSE GO TO CERR11
CLAB120NCA=NCA+2
1       IND=IND+2
1       GO TO C70      END
      IF FORMAL=0 THEN GOTO CLAMT20
CLAMT20
      IF IND=CL THEN GO TO CERR4
      NCA=NCA+1
      GO TO CLAB11
CLAMT200
      IF APOST=1 THEN GOTO CERR9
      IF INDEX('LU',SEP)=0 THEN GOTO CERR9
      IF ANAL(IND+1)=' ' THEN GOTO CERR9
      NCA=NCA+1
      IND=IND+1
      GOTO C101
C700   IF FORMAL =0 THEN GO TO C91
C1010  AUXOP3=SUBSTR(ZONE,I,NCA)
C740   I=IND+1

```

NEST

```

CLAB130 IF ANAL(I) =* * THEN GO TO CLAB14
      IF I=CL THEN DO
1      C730  OPERAND(PTLIB)=AUXOP3
1      C1210 PTLIB=PTLIB+1
1      GO TO C1      END
      I=I+1 GO TO CLAB13
      CLAB140
      IF INDEX(COMMENT,SUBSTR(ZONE,I,2)) =0 THEN
      DO
1      CEX550
      NCA=50
1      CLAMT110
1      IF ANAL3(NCA)=* * THEN DO
2      IF NCA,1 THEN NCA=NCA-1
2      ELSE GOTO CLAMT10
2      GOTO CLAMT11
2      END
1      CLAMT100
      SUBSTR(AUXOP3,NCA+2)=SUBSTR(ZONE,I,CL-I+1)
1      END
      ELSE GO TO CERR10
      GO TO C73
1      C710  IF FORMAL =0 THEN DO
      CAT=1 GO TO C92  END
      AUXOP2=SUBSTR(ZONE,I,NCA)
1      C950  I=IND+1
      IF ANAL(I) ='E' THEN
      IF FORMAL=0 THEN GOTO CERR5
      ELSE DO
1      INT=2      GOTO CLAMT5
1      END
      I=I+1
      IF ANAL(I) ='E' THEN DO
1      INT=1
1      I=I+1      END
      ELSE INT=0
      IF INDEX(LETTER,ANAL(I))=0 THEN GO TO CERR3
1      CLAMT50
      NCA=1
1      CLAB150 IND=I+NCA
      SEP=ANAL(IND)
      IF SEP=* * THEN
      IF APOST=1 THEN GO TO CERR9
      ELSE GO TO C80
      IF SEP=CATOP THEN GO TO CERR5
      IF SEP=*** THEN DO
1      IF ANAL(IND+1) ='H' THEN GO TO CLAB16
1      ELSE IF ANAL(IND+1) ='O' THEN GO TO CLAB16
1      ELSE GO TO CERR11
1      CLAB160 WORK=SUBSTR(ZONE,IND,2)
1      IND=IND+2
1      GO TO C80      END
      IF IND=CL THEN GO TO CERR4
      NCA=NCA+1
      GO TO CLAB15
1      C800  IF NCA,6 THEN DO
1      PUT SKIP
1      PUT LIST(WARN2,PRINT1,ATCARD,WARN3)
1      NCA=6      END
      RECORD=SUBSTR(ZONE,I,NCA)
      IF INT=1 THEN DO
1      SUBSTR(RECORD,2)=RECORD
1      ANAL2(I)='E'      END
      ELSE IF INT=0 THEN CALL SEARARG
      NCB= INDEX(AUXOP2,' ')
      IF NCB=0 THEN NCB=16
      K=INDEX(RECORD,' ')
      IF K=0 THEN K=7
      ELSE IF K,1 THEN K=K-1
      AUXOP3=SUBSTR(AUXOP2,1,NCB-1) CATOP SUBSTR(RECORD,1,K) WORK
      GO TO C74
1      C240  IF ANAL(IND+1) ='E' THEN GO TO CERR5
      SEP=ANAL(IND+2) IF SEP='E' THEN GO TO C30
      IF INDEX(LETTER,SEP)=0 THEN GO TO CERR3

```

```

      J=IND+2
C30A0 NCA=1
CLAB170 IND=J+NCA
      SEP=ANAL(IND)
      IF SEP=' ' THEN GO TO C25
      IF SEP=CATOP THEN GO TO CERR5
      IF SEP='=' THEN DO
1      LABIND=1 GO TO C25  END
      IF IND=CL THEN GO TO CERR4
      NCA=NCA+1
      GO TO CLAB17
C250  IF NCA,6 THEN DO
1      PUT SKIP
1      PUT LIST(WARN2,PRINT1,ATCARD,WARN3)
1      NCA=6      END
      AUXOP1=RECORD
      RECORD=SUBSTR(ZONE,J,NCA)
      IF INT=0 THEN CALL SEARARG
      ELSE IF INT=1 THEN DO
1      INT=0
1      SUBSTR(RECORD,2)=RECORD
1      ANAL2(1)='&'
1      END
      ELSE INT=0
      GO TO C15A
C300  J=IND+3
      IF INDEX(LETTER,ANAL(J))=0 THEN GO TO CERR3
      INT=1 GO TO C30A
C400  J=IND+1
      IF ANAL(J)='&' THEN DO
1      INT=2 GO TO CLAB18  END
      IF ANAL(J+1)='&' THEN DO
1      INT=0
1      J=J+1      END
      ELSE DO
1      INT=1
1      J=J+2  END
CLAB180 IF INDEX(LETTER,ANAL(J))=0 THEN GO TO CERR3
      GO TO C30A
C500  NCA=1
CLAB190 IND=I+NCA
      IF ANAL(IND)=' ' THEN GO TO C51
      IF ANAL(IND)=CATOP THEN DO
1      CAT=1 GO TO C51      END
      IF IND=CL THEN GO TO CERR4
      NCA=NCA+1
      GO TO CLAB19
C510  IF NCA,6 THEN DO
1      PUT SKIP
1      PUT LIST(WARN2,PRINT1,ATCARD,WARN3)
1      NCA=6      END
      RECORD=SUBSTR(ZONE,I,NCA)
      IF CAT =1 THEN DO I=IND+1
1      CODE(PTLIB)=RECORD
1      IF INDEX(STRING1,RECORD) =0 THEN GO TO CERR13
1      PNT=INDEX(STRING2,RECORD)
1      IF PNT=1 THEN GO TO CONDBRANCH
1      IF PNT=9 THEN GO TO UNCOBRANCH
1      PNT=INDEX(STRING3,RECORD)
1      IF PNT='0' THEN DO
2      IF INT =1 THEN GOTO CERR14
2      IF PNT=1 THEN GOTO SETARITM
2      ELSE GOTO SETCAR
2      END
1      IF RECORD='NOTE ' THEN GO TO CNOTE
1      GO TO C52  END
      IF ANAL(IND+1)='&' THEN GO TO CERR5
      IF ANAL(IND+2)='&' THEN DO
1      INT=1 I=IND+3  END
      ELSE DO INT=0 I=IND+2  END
      IF INDEX(LETTER,ANAL(I))=0 THEN GO TO CERR3
      NCA=1
CLAB200 IND=I+NCA
      SEP=ANAL(IND)

```

```

      IF SEP=' ' THEN GOTO CLAB21
      IF SEP=CATOP THEN GOTO CERR5
      IF IND=CL THEN GO TO CERR4
      NCA=NCA+1
      GO TO CLAB20
CLAB210 IF NCA,6 THEN DO
1       PUT SKIP
1       PUT LIST(WARN2,PRINT1,ATCARD,WARN3)
1       NCA=6      END
      AUXOP1=RECORD
      RECORD=SUBSTR(ZONE,I,NCA)
      IF INT=0 THEN CALL SEARARG
      ELSE DO
1       SUBSTR(RECORD,2)=RECORD
1       ANAL2(1)='&'
1       INT=0      END
      PNT=INDEX(AUXOP1,' ')
      IF PNT=0 THEN PNT=8
      CODE(PTLIB)=SUBSTR(AUXOP1,1,PNT-1)  CATOP  RECORD
      I=IND+1
C520   IF ANAL(I) = ' ' THEN GO TO C18
      IF I=CL THEN
      IF CODE(PTLIB)='MEND' THEN GO TO C23
          ELSE GO TO C121
      I=I+1
      GO TO C52
C600   I=I+1
      IF ANAL(I)='&' THEN DO
1       INT=1
1       I=I+1      END
      ELSE INT=0
      IF INDEX(LETTER,ANAL(I))=0 THEN GO TO CERR3
      NCA=1
CLAB220 IND=I+NCA
      SEP=ANAL(IND)
      IF SEP=' ' THEN GOTO C61
      IF SEP=CATOP THEN DO
1       CAT=1  GO TO C61      END
      IF IND=CL THEN GO TO CERR4
      NCA=NCA+1
      GO TO CLAB22
C610   IF NCA,6 THEN DO
1       PUT SKIP
1       PUT LIST(WARN2,PRINT1,ATCARD,WARN3)
1       NCA=6      END
      RECORD=SUBSTR(ZONE,I,NCA)
      IF INT=1 THEN DO
1       INT=0
1       SUBSTR(RECORD,2)=RECORD
1       ANAL2(1)='&'      END
      ELSE CALL SEARARG
      IF CAT=1 THEN GO TO C62
      CODE(PTLIB)=RECORD
      I=IND+1  GO TO C52
C620   IF ANAL(IND+1)='&' THEN DO
1       IF ANAL(IND+2)='&' THEN DO
2       INT=1  I=IND+3      END
1       ELSE I=IND+2      END
1       ELSE DO
1       INT=2
1       I=IND+1      END
      IF INDEX(LETTER,ANAL(I))=0 THEN GO TO CERR3
      NCA=1
CLAB230 IND=I+NCA
      SEP=ANAL(IND)
      IF SEP=' ' THEN GO TO C63
      IF SEP=CATOP THEN GO TO CERR5
      IF IND=CL THEN GO TO CERR4
      NCA=NCA+1
      GO TO CLAB23
C630   IF NCA,6 THEN DO
1       PUT SKIP
1       PUT LIST(WARN2,PRINT1,ATCARD,WARN3)

```

```

1      NCA=6      END
      AUXOP1=RECORD
      RECORD=SUBSTR(ZONE,I,NCA)
      IF INT=0 THEN DO
1      CALL SEARARG GO TO CLAB24  END
      IF INT=1 THEN DO
1      SUBSTR(RECORD,2)=RECORD
1      ANAL2(1)='&' END
      INT=0
CLAB240 I=IND+1
      IND=INDEX(AUXOP1,' ')
      IF IND=0 THEN IND=8
      CODE(PTLIB)=SUBSTR(AUXOP1,1,IND-1) CATOP RECORD
      CAT=0 GO TO C52
      C60
      SEP=ANAL(I+1)
      IF SEP=''''' THEN GOTO C100
      IF SEP='&' THEN DO
1      C900 APOST=1
1      GO TO C7  END
      IF ANAL(I+2)='&' THEN DO
1      J=I I=I+3
1      FORMAL=2  END
      ELSE DO
1      J=I I=I+2
1      FORMAL=1  END
      IF INDEX(LETTER,ANAL(I))=0 THEN GO TO CERR3
      ELSE GO TO C90
1      C910 NCA=NCA-2
1      C920 IF NCA,6 THEN DO
1      PUT SKIP
1      PUT LIST(WARN2,PRINT1,ATCARD,WARN3)
1      NCA=6      END
      RECORD=SUBSTR(ZONE,I,NCA)
      IF FORMAL=1 THEN CALL SEARARG
      ELSE DO
1      SUBSTR(RECORD,2)=RECORD
1      ANAL2(1)='&' END
      IF CAT=1 THEN DO
1      AUXOP2=''''' RECORD
1      GO TO C95  END
      ELSE DO
1      K=INDEX(RECORD,' ')
1      IF K=0 THEN K=7
1      ELSE IF K,1 THEN K=K-1
1      AUXOP3=''''' SUBSTR(RECORD,1,K) SUBSTR(ZONE,IND-2,2)
1      GO TO C74  END
1      C1000 J=I
      I=I+2
      IF ANAL(I)='&' THEN GO TO C104
      INT=2
      NCA=1
CLAB250 IND=I+NCA
      SEP=ANAL(IND)
      IF SEP=''''' THEN GO TO CLAB26
      IF SEP=CATOP THEN GO TO C102
      IF IND=CL THEN GO TO CERR4
CLAB270 NCA=NCA+1
      GO TO CLAB25
CLAB260 IF ANAL(IND+1)='&' THEN GO TO CLAB27
      I=J
      NCA=NCA+4
      IND=IND+1
      GO TO C101
1      C1020 IF ANAL(IND+1)='&' THEN GO TO CLAB28
      NCA=NCA+2
      AUXOP3=SUBSTR(ZONE,J,NCA)
      I=IND+2
1      C1070 IF ANAL(I)='&' THEN DO
1      INT=1
1      I=I+1      END
      ELSE INT=0
      IF INDEX(LETTER,ANAL(I))=0 THEN GO TO CERR3
1      C107A0

```

```

      NCB=1
CLAB290 IND=I+NCB
      SEP=ANAL(IND)
      IF SEP=**** THEN
        IF ANAL(IND+1)=**** THEN GO TO C103
        ELSE GO TO CERR15
      IF SEP=CATOP THEN GO TO CERR5
      IF IND=CL THEN GO TO CERR4
      NCB=NCB+1
      GO TO CLAB29
CLAB280
      IF INT=2 THEN GOTO CERR5
      I=IND+1
      INT=2
      GO TO C107A
C1030 RECORD=SUBSTR(ZONE,I,NCB)
      IF INT=1 THEN DO
1        SUBSTR(RECORD,2)=RECORD
1        ANAL2(1)='£' END
      ELSE
        IF INT=0 THEN CALL SEARARG
        WORK=*****
        INT=INDEX(RECORD,' ')
        IF INT=0 THEN INT=7
        ELSE IF INT,1 THEN INT=INT-1
        SUBSTR(AUXOP3,NCA+1,10)=CATOP SUBSTR(RECORD,1,INT) WORK
        IND=IND+2
        NCA=NCA+10
        GO TO C74
C1040 IF ANAL(I+1)='£' THEN DO
1        INT=1
1        I=I+2 END
      ELSE DO
1        I=I+1
1        INT=0
1        END
        IF INDEX(LETTER ,ANAL(I))=0 THEN GO TO CERR3
        NCA=1
CLAB300 IND=I+NCA
      SEP=ANAL(IND)
      IF SEP=**** THEN GO TO CLAB31
      IF SEP=CATOP THEN DO
1        CAT=1
1        GOTO CLAB32 END
        IF IND=CL THEN GO TO CERR4
        NCA=NCA+1
        GO TO CLAB30
CLAB310 IF ANAL(IND+1)=**** THEN GO TO CLAB32
        GO TO CERR15
CLAB320 RECORD=SUBSTR(ZONE,I,NCA)
      IF INT=1 THEN DO
1        SUBSTR(RECORD,2)=RECORD
1        ANAL2(1)='£' END
      ELSE CALL SEARARG
        WORK=*****
        IF CAT=1 THEN GOTO C106
        K=INDEX(RECORD,' ')
        IF K=0 THEN K=7
        ELSE K=K-1
        AUXOP3=WORK SUBSTR(RECORD,1,K) WORK
        NCA=11
        IND=IND+2
        GO TO C74
C1060 AUXOP3=WORK RECORD
      J=INDEX(RECORD,' ')
      IF J=0 THEN J=7
      ELSE J=J-1
      NCA=J+2
      GO TO C102
C1200 IF ANAL(I+1)='- ' THEN
      GOTO C1
      ELSE GOTO C7
C1500 J=1
      I=I+1

```

```

      IF ANAL(I)='&' THEN DO
1       I=I+1
1       IF ANAL(I)='&' THEN DO
2       INT=1 I=I+1           END
1       ELSE INT=0
1       IF INDEX(LETTER,ANAL(I))=0 THEN GO TO CERR3
1       END
      ELSE INT=2
      NCA=1
CLAB330 IND=I+NCA
      SEP=ANAL(IND)
      IF SEP='') THEN GO TO C151
      IF SEP=CATOP THEN DO
1      CAT=1 GO TO C151 END
      IF IND=CL THEN GO TO CERR4
      NCA=NCA+1
      GO TO CLAB33
C1510 IF INT=2 THEN DO
1      IF CAT=1 THEN DO
2      AUXOP2=SUBSTR(ZONE,I,NCA)
2      GO TO C153           END
1      NCA=NCA+2
1      AUXOP3=SUBSTR(ZONE,J,NCA)
1      GO TO C74           END
1      IF NCA,6 THEN DO PUT SKIP
1      PUT LIST(WARN2,PRINT1,ATCARD,WARN3)
1      NCA=6 END
      RECORD=SUBSTR(ZONE,I,NCA)
      IF INT=1 THEN DO
1      SUBSTR(RECORD,2)=RECORD
1      ANAL2(1)='&' END
      ELSE CALL SEARARG
      IF CAT=1 THEN DO
1      AUXOP2=RECORD GO TO C153 END
      K=INDEX(RECORD,' ')
      IF K=0 THEN K=7
      ELSE K=K-1
      AUXOP3=(' SUBSTR(RECORD,1,K) ')
      NCA=10
      GO TO C74
C1530 I=IND+1
      IF ANAL(I)='&' THEN GO TO CLAB34
      IF INT=2 THEN GO TO CERR5
      INT=2
      GO TO C152
CLAB340 I=I+1
1      IF ANAL(I)='&' THEN DO
1      INT=1 I=I+1           END
1      ELSE INT=0
1      IF INDEX(LETTER,ANAL(I))=0 THEN GO TO CERR3
C1520 NCA=1
CLAB350 IND=I+NCA
      SEP=ANAL(IND)
      IF SEP='') THEN GO TO CLAB36
      IF SEP=CATOP THEN GO TO CERR5
      IF IND=CL THEN GO TO CERR4
      NCA=NCA+1
      GO TO CLAB35
CLAB360 IF NCA,6 THEN DO
1      PUT SKIP
1      PUT LIST(WARN2,PRINT1,ATCARD,WARN3)
1      NCA=6           END
1      RECORD=SUBSTR(ZONE,I,NCA)
1      IF INT=0 THEN CALL SEARARG
1      ELSE
1      IF INT=1 THEN DO
1      SUBSTR(RECORD,2)=RECORD
1      ANAL2(1)='&' END
1      NCB=INDEX(AUXOP2,' ')
1      IF NCB=0 THEN NCB=16
      K=INDEX(RECORD,' ')
      IF K=0 THEN K=7
      ELSE IF K,1 THEN K=K-1
      AUXOP3=(' SUBSTR(AUXOP2,1,NCB-1) CATOP SUBSTR(RECORD,1,K) ')

```

```

      NCA=NCB+K+3
      GO TO C74
C180  CAT,INT=0
      NCB=1  SEP=ANAL(I)
      IF INDEX(COMMENT,SUBSTR(ZONE,I,2)) =0 THEN
DO
1      NCA=1
1      GOTO CLAMT10
1      END
      IF INDEX(DIGITS,SEP) =0 THEN GO TO C7
      IF INDEX(LETTER,SEP) =0 THEN GO TO C300
      IF INDEX('-',SEP) =0 THEN GO TO C120
      IF SEP=''' THEN GO TO C6
      IF SEP='&' THEN GO TO C200
      GO TO CERR12
C2000 I=I+1
      IF ANAL(I)='&' THEN DO
1      INT=1 I=I+1  END
      ELSE INT=0
      IF INDEX(LETTER,ANAL(I))=0 THEN GO TO CERR3
      GO TO CLAB37
C3000 INT=2
CLAB37ONCA=1
      TYPE=0
CLAB380IND=I+NCA
      SEPI=ANAL(IND)
      IF INDEX('+-',SEPI) =0 THEN GO TO C201
      IF SEPI=CATOP THEN
DO
1      IF CAT=0 THEN DO
2      CAT=1  GO TO C202  END
1      ELSE GO TO CERR5
1      END
      IF IND=CL THEN GO TO CERR4
      NCA=NCA+1
      GO TO CLAB38
C2010 CAT=0
C2020 IF INT =2 THEN GO TO CLAB39
      IF NCA=6 THEN GO TO CLAB40
      IF NCA=7 THEN DO
1      IF ANAL(IND-1)='.' THEN GO TO CLAB40
1      CLAB41ONCA=6
1      PUT SKIP
1      PUT LIST(WARN2,PRINT1,ATCARD,WARN3)
1      GO TO CLAB40  END
1      IF ANAL(IND-1)='.' THEN POINT=1
      GO TO CLAB41
      CLAB400 RECORD=SUBSTR(ZONE,I,NCA)
      IF POINT=1 THEN DO
1      ANAL2(7)='.'
1      POINT=0  END
      CLAB420AUXOP2=RECORD
      J=INDEX(AUXOP2,' ')-1
      IF J=-1 THEN J=7
      SUBSTR(AUXOP3,NCB,J)=RECORD
      NCB=NCB+J
      ANAL3(NCB)=SEPI
      NCB=NCB+1
      I=IND+1
CLAB430IF ANAL(I)=' ' THEN
      IF I=CL THEN DO
1      IF SEPI=' ' THEN GO TO C73
1      ELSE GO TO CERR10
1      END
      ELSE DO
1      I=I+1
1      GO TO CLAB43  END
      SEP=ANAL(I)
      IF SEP='&' THEN GO TO C200
      IF INDEX(COMMENT,SUBSTR(ZONE,I,2)) =0 THEN
      GOTO CEX55
CEXTRA20 IF INDEX(LETTER,SEP)=0 THEN
      IF INDEX(DIGITS,SEP)=0 THEN DO
1      IF TYPE =0 THEN GOTO CERR10

```

```

1      IF SEP1 =' ' THEN GOTO CERR10
1      IF INDEX('+-',SEP)=0 THEN GOTO CERR10
1      ANAL3(NCB-1)=SEP
1      CEXTRA30
1      I=I+1
1      IF ANAL(I)=' ' THEN
1      IF I=CL THEN GOTO CERR10
1      ELSE GOTO CEXTRA3
1      TYPE=1
1      SEP=ANAL(I)
1      GOTO CEXTRA2
1      END
1      IF CAT=1 THEN
1      IF INT=2 THEN GO TO CERR5      CEXTRA10 INT=2 GO TO CLAB37
1      CLAB390
1      IF NCA,6 THEN DO
1      PUT SKIP LIST(WARN2,PRINT1,ATCARD,WARN3)
1      NCA=6      END
1      RECORD=SUBSTR(ZONE,I,NCA)
1      IF INT=1 THEN DO
1      SUBSTR(RECORD,2)=RECORD
1      ANAL2(1)='&'      END
1      ELSE CALL SEARARG
1      GO TO CLAB42
1      UNCOBRANCH0 J=1
1      CEX200 IF ANAL(I)=' ' THEN DO
1      IF I=CL THEN GO TO CERR16
1      I=I+1 GO TO CEX20      END
1      IF ANAL(I)='.' THEN I=I+1
1      ELSE DO
1      PUT SKIP
1      PUT LIST(ERROR31,PRINT1,ATCARD,WARN5)
1      END
1      IF INDEX(LETTER,ANAL(I))=0 THEN GO TO CERR3
1      NCA=1
1      CLAB440 IND=I+NCA
1      IF ANAL(IND)=' ' THEN GO TO CEX1
1      IF IND=CL THEN GO TO CERR4
1      NCA=NCA+1
1      GO TO CLAB44
1      CEX10 IF NCA,6 THEN DO
1      PUT SKIP
1      PUT LIST(WARN2,PRINT1,ATCARD,PRINT2)      NCA=6      END
1      RECORD=SUBSTR(ZONE,I,NCA)      NCA=6
1      SUBSTR(AUXOP3,J,NCA)=RECORD
1      OPERAND(PTLIB)=AUXGP3
1      IF PTDEF,100 THEN DO
1      PUT SKIP EDIT(ERROR40,NOM1,PRINT1,ATCARD,PRINT2)
1      (A,X(1),A,X(1),A,F(4),X(1),A)
1      GO TO CERREX      END
1      INSPOINT(PTDEF)=J
1      I=IND+1
1      CLAB450 IF ANAL(I)=' ' THEN
1      IF I=CL THEN GO TO CEX4.
1      ELSE DO
1      I=I+1 GO TO CLAB45      END
1      IF INDEX(COMMENT,SUBSTR(ZONE,I,2))=0 THEN
1      SUBSTR(OPERAND(PTLIB),J+NCA+1)=SUBSTR(ZONE,I)
1      ELSE GO TO CERR10
1      CEX40 BRDEF(PTDEF)=PTLIB      PTDEF=PTDEF+1
1      GO TO C121
1      CONDBRANCH0 IF ANAL(I)=' ' THEN
1      IF I=CL THEN GO TO CERR17
1      ELSE DO
1      I=I+1 GO TO CONDBRANCH      END
1      BLANK=0
1      IF ANAL(I)='(' THEN GOTO CERR17
1      CLAB460 I=I+1
1      IF ANAL(I)=' ' THEN
1      IF I=CL THEN GO TO CERR17
1      ELSE GO TO CLAB46
1      IF ANAL(I)='&' THEN GO TO CEX12
1      I=I+1
1      TYPE=0

```

```

      IF ANAL(I)='£' THEN DO
1      INT=1  I=I+1  END
      ELSE INT=0
      IF INDEX(LETTER,ANAL(I))=0 THEN GO TO CERR3
      CEX130NCA=1
      CLAB470IND=I+NCA
      IF ANAL(IND)=' ' THEN GO TO CEX10
      IF IND=CL THEN GO TO CERR4
      NCA=NCA+1  GO TO CLAB47
      CEX100IF INT =2 THEN
      IF NCA,6 THEN DO
1      PUT SKIP
1      PUT LIST(WARN2,PRINT1,ATCARD,WARN3)
1      NCA=6  END
      RECORD=SUBSTR(ZONE,I,NCA)
      IF INT=0 THEN CALL SEARARG
      ELSE IF INT=1 THEN DO
1      SUBSTR(RECORD,2)=RECORD
1      ANAL2(1)='£'  END
      ANAL3(1)='('
      SUBSTR(AUXOP3,2,7)=RECORD
      I=IND+1
      CLAB480IF ANAL(I)=' ' THEN
      IF I=CL THEN GO TO CERR17
      ELSE DO
1      I=I+1
1      GO TO CLAB48  END
      ORDRE=SUBSTR(ZONE,I,2)
      IF INDEX(OPERA,ORDRE)=0 THEN GOTO CERR18
      J= INDEX(AUXOP3,' ')
      SUBSTR(AUXOP3,J+1,2)=ORDRE
      J=J+4
      GO TO CEX11
      CEX120
      TYPE=1
      IF INDEX(DIGITS,ANAL(I))=0 THEN
      IF INDEX(LETTER,ANAL(I))=0 THEN GOTO CERR3
      INT=2
      GO TO CEX13
      CEX110I=I+3
      IF ANAL(I) ='£' THEN GO TO CLAB49
      I=I+1
      IF ANAL(I)='£' THEN DO
1      INT=1  I=I+1      END
      ELSE INT=0
      CEX150NCA=1
      IF INDEX(LETTER,ANAL(I))=0 THEN GOTO CERR3
      CLAB500 IND=I+NCA
      SEP=ANAL(IND)
      IF SEP=' ' THEN DO
1      BLANK=1  GO TO CEX14  END
      IF SEP=')' THEN GO TO CEX14
      IF IND=CL THEN GO TO CERR4
      NCA=NCA+1
      GO TO CLAB50
      CLAB490 IF TYPE=1 THEN GO TO  CERR19
      INT=2
      SEP=ANAL(I)
      IF INDEX(LETTER,SEP)=0 THEN IF INDEX(DIGITS,SEP)=0 THEN DO
1      IF SEP ='''' THEN GO TO CERR17
1      IF ANAL(I+2) ='''' THEN GO TO CERR17
1      SEP1=ANAL(I+3)
1      IF SEP1=' ' THEN BLANK=1
1      ELSE
1      IF SEP1 =')' THEN GOTO CERR17
1      RECORD=SUBSTR(ZONE,I,3)
1      INT =3
1      IND=I+3
1      GO TO CCEX11  END
1      GO TO CEX15
      CEX140
      IF INT=2 THEN GO TO CLAB51
      IF NCA,6 THEN DO
1      PUT SKIP

```

```

1      PUT LIST(WARN2,PRINT1,ATCARD,PRINT2)
1      NCA=6      END
CLAB510RECORD=SUBSTR(ZONE,I,NCA)
      IF INT=0 THEN CALL SEARARG
      ELSE IF INT=1 THEN DO
1      SUBSTR(RECORD,2)=RECORD
1      ANAL2(1)='S'      END
      INT=INDEX(RECORD,' ')
      IF INT=0 THEN INT=8  INT=INT-1
CCEX110SUBSTR(AUXOP3,J,INT) = SUBSTR(RECORD,1,INT)
      IF BLANK =1 THEN GO TO CLAB52
      I=IND+1
CLAB530
      IF ANAL(I)=' ' THEN
      IF I=CL THEN GO TO CERR17
      ELSE DO
1      I=I+1 GO TO CLAB53  END
      IF ANAL(I)=')' THEN GO TO CLAB54
      ELSE GO TO CERR17
CLAB520I=IND
CLAB540ANAL3(J+INT)=')'
      I=I+1
      J=J+INT+1
      GO TO CEX20
INTDECLOJ=1
      BLANK,FIN=0
      PARAM=0
      CLAB550IF INDEX(LETTER,ANAL(I))=0 THEN GO TO CERR3
      NCA=1
CLAB560IND=I+NCA
      SEP=ANAL(IND)
      IF SEP=' ' THEN DO
1      BLANK=1
1      GO TO CLAB57      END
      IF SEP=', ' THEN GO TO CLAB57
      IF IND=CL THEN DO
1      NCA=NCA+1
1      FIN=1
1      GO TO CLAB57      END
      NCA=NCA+1 GO TO CLAB56
CLAB570IF NCA,6 THEN DO
1      PUT SKIP
1      PUT LIST(WARN2,PRINT1,ATCARD,WARN3)
1      NCA=6
1      END
      RECORD=SUBSTR(ZONE,I,NCA)
      CEX210SUBSTR(AUXOP3,J,NCA)=RECORD
      IF FIN=1 THEN GO TO C73
      I=IND+1
      IF BLANK=1 THEN DO
1      CLAB590IF ANAL(I) = ' ' THEN DO
2      BLANK=0
      GO TO CLAB58      END
1      IF I=CL THEN GO TO C73
1      I=I+1
1      GO TO CLAB59      END
1      IF I=CL THEN DO
1      FIN=1  NCA=1
1      IF INDEX(LETTER,ANAL(I))=0 THEN GO TO CERR3
1      ELSE GO TO CEX21  END
1      PARAM=PARAM+1
      IF PARAM,7 THEN GO TO CERR20
      GO TO CLAB60
CLAB580
      IF ANAL(I) =', ' THEN DO
1      IF INDEX(COMMENT,SUBSTR(ZONE,I,2)) =0 THEN
1      GOTO CLAB62A
1      GOTO CERR22      END
CLAB600
      ANAL3(J+NCA)=' ', '
      J=J+NCA+1
CLAB610
      IF ANAL(I) = ' ' THEN GO TO CLAB62
      IF I=CL THEN GO TO CERR23

```

```

      I=I+1
      GO TO CLAB61
CLAB620 IF ANAL(I)='- ' THEN DO
1      CLAB62A0
      SUBSTR(AUXOP3,J)=SUBSTR(ZONE,I)
1      GO TO C73          END
      GOTO CLAB55
INO
      CODE(PTLIB)=' '
      SUBSTR(AUXOP3,1,2)='I '
      PARAM=1
      J=3
CEX300IF ANAL(I)='£' THEN DO
1      I=I+1
1      IF ANAL(I)='£' THEN INT=0
1      ELSE DO
2          INT=1
2          I=I+1
2          END
1      IF INDEX(LETTER,ANAL(I))=0 THEN GOTO CERR3
1      END
      ELSE IF INDEX(DIGITS,ANAL(I))=0 THEN GOTO CERR21
      ELSE INT=2
      NCA=1
CLABG10
      IND=I+NCA
      SEP=ANAL(IND)
      IF SEP=CATOP THEN GOTO CERR5
      IF SEP='+' THEN
          IF IND=CL THEN GOTO CERR4
      ELSE DO
1          NCA=NCA+1
1          GOTO CLABG1
1      END
      IF INT=2 THEN GOTO CEX31
      IF NCA,6 THEN DO
1          PUT SKIP LIST(WARN2,PRINT1,ATCARD,WARN3)
1          NCA=6
1          END
CEX310
      RECORD=SUBSTR(ZONE,I,NCA)
      IF INT=0 THEN CALL SEARARG
      ELSE IF INT=1 THEN DO
1          SUBSTR(RECORD,2)=RECORD
1          ANAL2(1)='£'
1          END
      NCA=INDEX(RECORD,' ')
      IF NCA=0 THEN NCA=8
      SUBSTR(AUXOP3,J,NCA-1)=RECORD
      ANAL3(J+NCA-1)='+'
      J=J+NCA
      I=IND+1
      IF PARAM=1 THEN DO
1          PARAM=0
1          GOTO CEX30
1          END
      IF ANAL(I)='£' THEN GOTO CEX32
      I=I+1
      IF ANAL(I)='£' THEN DO
1          INT=1
1          I=I+1
1          END
      ELSE INT=0
      NCA=1
CLABG20
      IND=I+NCA
      SEP=ANAL(IND)
      IF SEP=CATOP THEN GOTO CERR5
      IF SEP=' '
          THEN IF IND=CL
              THEN DO
1          NCA=NCA+1
1          GOTO CLABG2
1          END

```

```

      ELSE GOTO CERR4
      IF NCA,6 THEN DO
1      PUT SKIP LIST(WARN2,PRINT1,ATCARD,WARN3)
1      NCA=6
1      END
      RECORD=SUBSTR(ZONE,I,NCA)
      IF INT=0 THEN CALL SEARARG
      ELSE DO
1      SUBSTR(RECORD,2)=RECORD
1      ANAL2(1)='E'
1      END
      NCA=INDEX(RECORD,' ')
      IF NCA=0 THEN NCA=8
      SUBSTR(AUXOP3,J,NCA)=RECORD
      J=J+NCA
CEX320
      SUBSTR(AUXOP3,J)=SUBSTR(ZONE,IND+1,CL-IND)
      GOTO C73
      BLANK=0
      IF ANAL(1) = '(' THEN GO TO CERR17
SETARITMO
      J=1
CEX520
      DO I=1 BY 1 WHILE(ANAL(I)=' ')
1      IF I=CL THEN GO TO CERR14
1      END
      SEPI=ANAL(I)
      IF SEPI='E' THEN DO
1      I=I+1
1      IF ANAL(I)='E' THEN DO
2      INT=1
2      I=I+1
2      END
1      ELSE INT=0
1      IF INDEX(LETTER,ANAL(I))=0 THEN GOTO CERR3
1      NCA=1
1      CLABG40
      IND=I+NCA
1      SEP=ANAL(IND)
1      IF INDEX(' */+-',SEP)=0 THEN
1      IF IND=CL THEN GOTO CERR4
1      ELSE DO
2      NCA=NCA+I
2      GOTO CLABG4
2      END
1      GOTO CEX50
1      END
      ELSE
      IF INDEX(DIGITS,SEPI)=0 THEN
      IF INDEX('+-',SEPI)=0 THEN GOTO CERR14
      NCA=0
      CLABG50
      NCA=NCA+1 IND=I+NCA
      SEP=ANAL(IND)
      IF INDEX(' */+-',SEP)=0 THEN
      IF INDEX(DIGITS,SEP)=0 THEN GOTO CERR14
      ELSE IF IND=CL THEN GOTO CERR4
      ELSE GOTO CLABG5
      RECORD=SUBSTR(ZONE,I,NCA)
      GOTO CEX51
CEX500
      IF NCA,6 THEN DO
1      PUT SKIP LIST(WARN2,PRINT1,ATCARD,WARN3)
1      NCA=6
1      END
      RECORD=SUBSTR(ZONE,I,NCA)
      IF INT=0 THEN DO
1      NCA=3
1      CALL SEARARG END
1      ELSE DO
1      SUBSTR(RECORD,2)=RECORD
1      ANAL2(1)='E'
1      NCA=NCA+I
1      END

```

```

CEX510
  I=IND+1
  SUBSTR(AUXOP3,J,NCA)=RECORD
  J=J+NCA
  IF SEP = ' ' THEN GO TO CEX63
DO WHILE(ANAL(I)=' ')
1   IF I=CL THEN GO TO C73
1   I=I+1  END
  SEP=ANAL(I)
  IF INDEX(COMMENT,SUBSTR(ZONE,I,2)) = 0 THEN
    GOTO CEX55
  INT=INDEX('*/+-',SEP)
  IF INT=0 THEN GO TO CERR14
  I=I+1
  CEX630
  ANAL3(J)=SEP
  J=J+1  GO TO CEX52
SETCARO
  CAT=0
  FIN=0
  K=1
CEX620
  IF ANAL(I) = ' ' THEN GO TO CEX60A
  ELSE IF I =CL THEN DO
1     I=I+1
1     GOTO CEX62
1     END
    ELSE IF FIN=0  SEP=CATOP THEN GOTO CERR14
  ELSE
    GO TO C73
CEX60A0
  IF INDEX(COMMENT,SUBSTR(ZONE,I,2))=0
  THEN GOTO CEX60
  IF FIN=0 THEN GOTO CERR14
  SUBSTR(AUXOP3,K)=SUBSTR(ZONE,I)
  GO TO C73
CEX600
  IF ANAL(I)='&' THEN DO
1     I=I+1
1     IF ANAL(I)='&' THEN DO
2         INT=1
2         I=I+1
2         END
1     ELSE INT=0
1     GOTO CLAMT40  END
  IF SUBSTR(ZONE,I,4)='MOVE' THEN GOTO CEX64
  INT=2
  IF ANAL(I)=''' THEN GO TO CEX61A
  GOTO CEX61AB
CEX61A0
  IF ANAL(I+2) =''' THEN GO TO CERR14
  SUBSTR(AUXOP3,K,3)=SUBSTR(ZONE,I,3)
  SEP=ANAL(I+3)
  ANAL3(K+3)=SEP
  IF SEP=CATOP THEN DO
1     IF CAT=1 THEN GOTO CERR5
1     CAT=1  END
  I=I+4
  K=K+4
  GO TO CEX62A
CLAMT400
  IF INDEX(LETTER,ANAL(I))=0 THEN GOTO CERR3
CEX61AB0
  NCA=1
CEX610
  IND=I+NCA
  SEP=ANAL(IND)
  IF SEP=' ' THEN GOTO CLAMT41
  IF SEP=CATOP THEN DO
1     IF CAT=1 THEN GOTO CERR5
1     CAT=1  GOTO CLAMT41
1     END
  IF IND=CL THEN GOTO CERR4
    NCA=NCA+1

```

```

                                GOTO CEX61
CLAMT410
    IF INT =2 THEN
        IF NCA,6 THEN DO
1          PUT SKIP LIST(WARN2,PRINT1,ATCARD,WARN3)
1          NCA=6  END
        RECORD=SUBSTR(ZONE,I,NCA)
        IF INT=0 THEN CALL SEARARG
        ELSE IF INT=1 THEN DO
1          SUBSTR(RECORD,2)=RECORD
1          ANAL2(1)='&'
1          END
        J=INDEX(RECORD,' ')-1
        IF J=-1 THEN J=7
        SUBSTR(AUXOP3,K,J)=RECORD
        ANAL3(K+J)=SEP
        I=IND+1
        K=K+J+1
    CEX62A0
    IF FIN =0 THEN GOTO CLAB13
        FIN=1
        GO TO CEX62
    CEX640
        J=1
        I=I+4
        IF ANAL(I)='('
            THEN DO
1          K=1
1          SEP=', '
1          SUBSTR(AUXOP3,J,5)=SUBSTR(ZONE,I-4,5)
1          J=J+5
1          I=I+1
    CEX650 IF K=4 THEN SEP='')'
1          IF ANAL(I)='&'
            THEN DO
2          I=I+1
2          IF ANAL(I)='&'
            THEN DO
3          I=I+1
3          INT=1
3          END
2          ELSE INT=0  END
1          ELSE DO
2          INT=2
2          GOTO CLAMT30
2          END
1          IF INDEX(LETTER,ANAL(I))=0 THEN GOTO CERR3
1          CLAMT300
            NCA=1
1          CLAMT30A0
            IND=I+NCA
1          IF ANAL(IND)=SEP THEN GOTO CLAMT31
1          IF IND=CL THEN GOTO CERR24
1          NCA=NCA+1
1          GOTO CLAMT30A
1          CLAMT310
            IF INT =2
            THEN IF NCA,6 THEN DO
2          PUT SKIP LIST(WARN2,PRINT1,ATCARD,WARN3)
2          NCA=6
2          END
1          RECORD=SUBSTR(ZONE,I,NCA)
1          IF INT=0 THEN CALL SEARARG
1          ELSE IF INT=1 THEN DO
2          SUBSTR(RECORD,2)=RECORD
2          ANAL2(1)='&'
2          END
1          INT=INDEX(RECORD,' ')
1          IF INT=0 THEN INT=8
1          SUBSTR(AUXOP3,J,INT-1)=RECORD
1          J=J+INT-1
1          ANAL3(J)=SEP
1          K=K+1
1          I=IND+1

```

NEST

```

1          J=J+1
1      IF K,4 THEN GOTO CLAB13
1      ELSE GOTO CEX65
1      END
      ELSE GOTO CERR24
INBRANCHO
    J=1
PASCUAL10
    I=BRDEF(J)
    K=INSPPOINT(J)
    NOM=SUBSTR(OPERAND(I),K,6)
    PNT=1
PASCUAL20
    IF NOM=LABNAME(PNT) THEN GOTO EXPASC1
    IF PNT=PTLAB THEN DO
1 PUT SKIP EDIT(PRINT5,NOM,PRINT20A,PRINT1,I-ADDR,PRINT7,NOM1,ERROR42,PRINT6)
    (A,X(1),A,X(1),A,X(1),A,F(5),X(1),A,X(1),A,X(1),A,X(1),A)
1      PNT=1
1      NOPUNCH=2
1      GOTO PASCUAL3
1      END
      PNT=PNT+1
      GO TO PASCUAL2
EXPASC10
    PNT=LABADDR(PNT)
    PNT=PNT-I
PASCUAL30
    SUBSTR(OPERAND(I),42,9)=PNT
    J=J+1
    IF J=PTDEF THEN GOTO RETURPT
    ELSE GOTO PASCUAL1
NESTINGO
    J=I
    NEST=1
    ARGUM(*,2)=ARGUM(*,1)
    PTARGT=PTARG
    MCCALL=1 ERRIND=0
    CALL MACANAL
    NEST=0
    IF ERRIND=1 THEN GOTO C1
    ARGUM1=ARGUM(*,1)
    PTARG1=PTARG
    PTARG=PTARGT
    ARGUM(*,1)=ARGUM(*,2)
    ARGUM(*,2)=' '
    IF PTARG1 =0 THEN GOTO NESTLAB1
    SUBSTR(OPERAND(PTLIB),2,2)='/'
NESTLAB20
    GOTO C121
NESTLAB10
    K=1
    J=1
PASCUAL40
    RECORD=ARGUM1(K)
    IF ANAL2(1)='&'
    THEN DO
1      RECORD=SUBSTR(RECORD,2)
1      IF ANAL2(1) ='&' THEN
1      CALL SEARARG
1      END
      IND=INDEX(RECORD,' ')
    IF IND=0 THEN IND=7
    ELSE IF IND,1 THEN
      IND=IND-1
      IF J,=44 THEN GOTO NESTLAB3
      SUBSTR(AUXOP3,J,IND)=RECORD
NESTLAB50
      ANAL3(J+IND)=' '
      J=J+IND+1
      IF K =PTARG1
      THEN DO
1 NESTLAB40
      K=K+1

```

SZ/TM

SZ/TM

```

1          GOTO PASCUAL4
1          END
          IF J,50
            THEN SUBSTR(CODE(PTLIB),15)='/'
            ELSE ANAL3(J)='/'
          OPERAND(PTLIB)=AUXOP3
          GOTO C121
NESTLAB30
          IF J+IND,50
            THEN DO
1          IF J)=50 THEN SUBSTR(AUXOP3,J)=',,,'
1          J=1
1          OPERAND(PTLIB)=AUXOP3
1          AUXOP3=' '
1          PTLIB=PTLIB+1
1          SUBSTR(AUXOP3,J,IND)=RECORD
1          GOTO NESTLAB5
1          END
          SUBSTR(AUXOP3,J,IND)=RECORD
          IF J+IND =50 THEN SUBSTR(AUXOP3,J+IND)=',,,'
ELSE ANAL3(50)=','
          OPERAND(PTLIB)=AUXOP3
          AUXOP3=' '
          PTLIB=PTLIB+1
          IF K =PTARG1
            THEN DO
1          J=1
1          GOTO NESTLAB4
1          END
          IF J+IND,=50
            THEN SUBSTR(CODE(PTLIB-1),15)='/'
            ELSE SUBSTR(OPERAND(PTLIB-1),J+IND,2)='/,/'
          GOTO C1
CNOTE0
          OPERAND(PTLIB)=SUBSTR(ZONE,1,CL-1+1)
          GO TO C121
CERR20PUT SKIP
          PUT LIST(ERROR1,PRINT1,ATCARD,PRINT2)
          GO TO CERREX
CERR30PUT SKIP
          PUT LIST(ERROR9,PRINT1,ATCARD,PRINT2)
          GO TO CERREX
CERR40PUT SKIP
          PUT LIST(ERROR2,PRINT1,ATCARD,PRINT2)
          GO TO CERREX
CERR50PUT SKIP
          PUT LIST(ERROR24,PRINT1,ATCARD,PRINT2)
          GO TO CERREX
CERR60PUT SKIP
          PUT LIST(ERROR3,PRINT1,ATCARD,PRINT2)
          GO TO CERREX
CERR70PUT SKIP
          PUT LIST(ERROR7,PRINT1,ATCARD,PRINT2)
          GO TO CERREX
CERR80PUT SKIP
          PUT LIST(ERROR1,PRINT3,PRINT1,ATCARD,PRINT2)
          GO TO CERREX
CERR90PUT SKIP
          PUT LIST(ERROR25,PRINT1,ATCARD,PRINT2)
          GO TO CERREX
CERR100 PUT SKIP
          PUT LIST(ERROR26,PRINT1,ATCARD,WARN4)
          GO TO C73
CERR110 PUT SKIP
          PUT LIST(ERROR27,PRINT1,ATCARD,PRINT2)
          GO TO CERREX
CERR120 PUT SKIP
          PUT LIST(ERROR1,PRINT4,PRINT1,ATCARD,PRINT2)
          GO TO CERREX
CERR130 PUT SKIP
          PUT LIST(ERROR28,PRINT1,ATCARD,PRINT2)
          GO TO CERREX
CERR140 PUT SKIP
          PUT LIST(ERROR29,PRINT1,ATCARD,PRINT2)

```

SZ/TP

SZ/TP

SZ/TP

SZ/TP

```

      GO TO CERREX
CERR150 PUT SKIP
        PUT LIST(ERROR30,PRINT1,ATCARD,PRINT2)
      GO TO CERREX
CERR160 PUT SKIP
        PUT LIST(ERROR32,PRINT1,ATCARD,PRINT2)
      GO TO CERREX
CERR170 PUT SKIP
        PUT LIST(ERROR33,PRINT1,ATCARD,PRINT2)
      GO TO CERREX
CERR180 PUT SKIP
        PUT LIST(ERROR34,PRINT1,ATCARD,PRINT2)
      GO TO CERREX
CERR190 PUT SKIP
        PUT LIST(ERROR35,PRINT1,ATCARD,PRINT2)
      GO TO CERREX
CERR200 PUT SKIP
        PUT LIST(ERROR36,PRINT1,ATCARD,PRINT2)
      GO TO CERREX
CERR210 PUT SKIP
        PUT LIST(ERROR37,PRINT1,ATCARD,PRINT2)
      GO TO CERREX
CERR220 PUT SKIP
        PUT LIST(ERROR16,PRINT1,ATCARD,PRINT2)
      GO TO CERREX
CERR230 PUT SKIP
        PUT LIST(ERROR15,PRINT1,ATCARD,PRINT2)
      GO TO CERREX
CERR240 PUT SKIP
        PUT LIST(ERROR39,PRINT1,ATCARD,PRINT2)
      GOTO CERREX
CERR250
  PUT SKIP EDIT(ERROR43,PRINT5,RECORD,PRINT1,ATCARD,PRINT2)
    (A,X(1),A,X(1),A,X(1),A,F(4),X(1),A)
  CERREX0
    LIBRARY1(PTLIB)=' '
    IF BRANCH=1 THEN DO
1      PTLAB=PTLAB-1
1      LABNAME(PTLAB)=' '
1      END
      NOPUNCH=2
      GO TO C1
SEARARGO
  PROCEDURE
    DO PNT=0 BY 1 WHILE(ARGUM(PNT,1) =RECORD)
1      IF PNT=PTARG THEN DO
2      PUT SKIP LIST(ERROR41,PRINT1,ATCARD,PRINT2)
2      GO TO CERREX
2      END
1      END
      ORDRE=NUMERICS(PNT)
      RECORD='A' ORDRE
      END
      END

```

CALLLIB0

PROCEDURE

/*DECLARATIONS*/

DCL SORTIE FILE STREAM

DCL DIGITS CHAR(10) INIT('0123456789') EXTERNAL

DCL 1 DIRECTG(20) EXTERNAL,

2 NAME CHAR(6) INIT((20)(6)' '),

2 ADRESSE FIXED BINARY INIT((20)0),

2 SWITCH FIXED BINARY INIT((20)0)

DCL 1 LIBRARY(500) EXTERNAL,

2 LABEL CHAR(15) INIT((500)(15)' '),

2 CODE CHAR(15) INIT((500)(15)' '),

2 OPERAND CHAR(50) INIT((500)(50)' ')

DCL LIBRARY1(500) CHAR(80) DEFINED LIBRARY

DCL ZONE1 CHAR(80) EXTERNAL

DCL ANAL1(80) CHAR(1) DEFINED ZONE1

DCL LETTER CHAR(26) INIT('ABCDEFGHIJKLMNOPQRSTUVWXYZ') EXTERNAL

DCL RECORD CHAR(7) EXTERNAL

DCL ANAL2(7) CHAR(1) DEFINED RECORD

DCL PTLIB FIXED BINARY(31) INIT(1) EXTERNAL

DCL ACTMAC CHAR(210) INIT((30)' *') EXTERNAL

DCL DYNJURE(20) FIXED BINARY EXTERNAL

DCL PTDYN FIXED BINARY INIT(1) EXTERNAL

DCL PTREC FIXED BINARY

DCL PTACT FIXED BINARY INIT(1) EXTERNAL

DCL EXTRN FIXED BINARY EXTERNAL

DCL CONDSTACK(20) FIXED BINARY

DCL PTCOND FIXED BINARY

DCL RESULT FIXED BINARY INIT(0)

DCL PTDIR FIXED BINARY INIT(1) EXTERNAL

DCL (I,J,K,NCA) FIXED BINARY EXTERNAL

DCL PTOUT FIXED BINARY(31) INIT(1) EXTERNAL

DCL END1 FIXED BINARY INIT(0)

DCL NOM CHAR(6) EXTERNAL

DCL ARGUM(0015,10) CHAR(7) EXTERNAL

DCL (IND,POINT) FIXED BINARY EXTERNAL

DCL CATOP CHAR(1) INIT('8') EXTERNAL

DCL SEP CHAR(1) EXTERNAL

DCL CAT FIXED BINARY EXTERNAL

DCL INT FIXED BINARY EXTERNAL

DCL AUXOP2 CHAR(15) EXTERNAL

DCL ANAL4(15) CHAR(1) DEFINED AUXOP2

DCL AUXOP1 CHAR(7) EXTERNAL

DCL AUXOP3 CHAR(50) EXTERNAL

DCL ANAL3(50) CHAR(1) DEFINED AUXOP3

DCL PNT FIXED BINARY EXTERNAL

DCL STRING1 CHAR(31) INIT('DCLA *DCGA *DCLC *DCGC ')

EXTERNAL

DCL NOPUNCH FIXED BINARY INIT(0) EXTERNAL

DCL ADDR FIXED BINARY EXTERNAL

DCL TYPE FIXED BINARY EXTERNAL

DCL OPERA CHAR(23) INIT('EQ*NE*GT*GE*LT*LE*NG*NL') EXTERNAL

DCL 1 GLOCHAR(100) EXTERNAL,

2 IDGCHAR CHAR(6) INIT((100)(6)' '),

2 VAGCHAR CHAR(15) INIT((100)(15)' ')

DCL 1 GLOARIT(110) EXTERNAL,

2 IDGOAR CHAR(6) INIT((110)(6)' '),

2 VAGOAR FIXED BINARY(31) INIT((110)0)

DCL PTGBAR FIXED BINARY INIT(1) EXTERNAL

DCL ORDRE CHAR(2)

DCL STRING2 CHAR(9) INIT('SETA*SETC')

DCL 1 LOCALCHAR(30),

2 IDLOCHAR CHAR(6),

2 VALOCHAR CHAR(15)

DCL SEP1 CHAR(1)

DCL AUXOP4 CHAR(15)

DCL ANAL5(15) CHAR(1) DEFINED AUXOP4

DCL PTGBCH FIXED BINARY INIT(1) EXTERNAL

DCL PTLOCHAR1 FIXED BINARY /*LOCAL CHARACTER POINTER*/

DCL FIN FIXED BINARY

DCL 1 LOCALARITM(30),

2 IDLOAR CHAR(6),

2 VALOAR FIXED BINARY(31)

DCL PTLOAR1 FIXED BINARY /*LOCAL ARITH POINT*/

DCL (PTLOCHAR2,PTLOAR2) FIXED BINARY

DCL NESTMATRIX(0015) FIXED BINARY /*NESTING MATRIX*/

Z/MB.

Z/MB.

Z/MB.

```

DCL INTERNAL CHAR(6)
DCL DUMPING FIXED BINARY INIT(0) EXTERNAL
DCL INTCHAR CHAR(15), ANALCHAR(15) CHAR(1) DEFINED INTCHAR
DCL INTARIT FIXED BINARY(31)
DCL OPCODE CHAR(4)
DCL CONVERT CHAR(14), ANALCONV(14) CHAR(1) DEFINED CONVERT
DCL INTARIT1 FIXED BINARY(31)
DCL TYPE1 FIXED BINARY
DCL REVERT FIXED BINARY
DCL PID FIXED BINARY
DCL SWITCH1 FIXED BINARY
DCL TYPEAUX FIXED BINARY
DCL INTARIT2 FIXED BINARY(31)
DCL MOVEOP(3) FIXED BINARY
DCL NESTSTACK(10) FIXED BINARY
DCL NESTSCOPE(40) FIXED BINARY
DCL (PTSTACK,PTSCOPE) FIXED BINARY
DCL PTDIR1 FIXED BINARY EXTERNAL
DCL FIELD FIXED BINARY
DCL FIND FIXED BINARY
DCL PTAUX FIXED BINARY
DCL 1 NESTDIR(10),
    2 NESTNAME CHAR(6),
    2 NESTADDR FIXED BINARY
DCL NESTLAB FIXED BINARY INIT(0)
DCL NOM1 CHAR(6)
DCL NESTERR FIXED BINARY INIT(0)
DCL LABARIT(205) LABEL
DCL DCLLAB(4) LABEL
DCL OPLAB(8) LABEL INIT(EQ,NE,GT,GE,LT,LE,NG,NL)
DCL AUXLAB1(003) LABEL INIT(BR7,BR14,BR25,BR27)
DCL AUXLAB3(003) LABEL INIT(BR3,BR17,BRER1,BRER2)
DCL AUXLAB4(003) LABEL INIT(BR4,BR19,BRER1,BRER2)
DCL COMMENT CHAR(5) INIT('--, ') EXTERNAL
DCL INTLABEL(4) LABEL INIT(DCLA,DCGA,DCLC,DCCG)
DCL WARN4 CHAR(19) INIT('IT HAS BEEN IGNORED') EXTERNAL
DCL PRINT00 CHAR(17)
DCL PRINT1 CHAR(19) INIT('IN STATEMENT NUMBER') EXTERNAL
DCL PRINT7 CHAR(8) INIT('OF MACRO') EXTERNAL
DCL PRINT8 CHAR(23) INIT('AND A SPECIAL CHARACTER')
DCL PRINT9 CHAR(26) INIT('IMPLEMENTATION RESTRICTION')
DCL PRINT12 CHAR(15) INIT('LOCAL CHARACTER')
DCL PRINT13 CHAR(16) INIT('LOCAL ARITHMETIC')
DCL PRINT14 CHAR(16) INIT('GLOBAL CHARACTER')
DCL PRINT15 CHAR(17) INIT('GLOBAL ARITHMETIC')
DCL PRINT16 CHAR(26) INIT('NESTING LEVEL EXCEEDING 10')
DCL PRINT18 CHAR(18) INIT('NUMBER OF INTERNAL')
DCL PRINT19 CHAR(21) INIT('THE INTERNAL VARIABLE')
DCL PRINT20 CHAR(32) INIT('THE INSTRUCTION HAS BEEN SKIPPED')
DCL PRINT21 CHAR(20) INIT('DOUBLE DEFINITION OF')
DCL PRINT22 CHAR(43) INIT('CONFLICTING DECLARATION OF INTERNAL
VARIABLE')
DCL PRINT23 CHAR(31) INIT('VARIABLES DEFINITION, EXCEEDS 30')
DCL ERROR42 CHAR(20) INIT('HAS NOT BEEN DEFINED') EXTERNAL
DCL ERROR43 CHAR(29) INIT('ERROR IN LIBRARY ORGANIZATION')
DCL ERROR44 CHAR(43) INIT('INVALID USE OF ARITHMETIC INTERNAL V
ARIABLE')
DCL ERROR45 CHAR(42) INIT('INVALID USE OF CHARACTER INTERNAL VA
RIABLE')
DCL ERROR46 CHAR(29) INIT('INVALID ARITHMETIC COMPARISON')
DCL ERROR47 CHAR(45) INIT('INVALID COMPARISON BETWEEN INTERNAL
VARIABLES')
DCL ERROR48 CHAR(48) INIT('INVALID COMPARISON INVOLVING A SPECI
AL CHARACTER')
DCL ERROR49 CHAR(57) INIT('INTERNAL CHARACTER VARIABLE AS TARGE
T IN SETA INSTRUCTION')
DCL ERROR50 CHAR(58) INIT('INTERNAL ARITHMETIC VARIABLE AS TARG
ET IN SETC INSTRUCTION')
DCL ERROR51 CHAR(59) INIT('ARITHMETIC VARIABLE AS THE SOURCE ST
RING IN A MOVE FUNCTION')
DCL ERROR52 CHAR(59) INIT('CHARACTER VARIABLE AS POSITIONAL OPE
RAND IN A MOVE FUNCTION')
DCL ERROR53 CHAR(30) INIT('UNDEFINED MACRO IN NESTED CALL')
ON ERROR BEGIN

```

2/MB.

```

DCL UNIT CHAR(8)
UNIT='CALLLIB'
PUT SKIP EDIT('AN ERROR OCCURRED IN PROC',UNIT)
      (A,X(1),A)
PUT SKIP EDIT('INDICATIVE DATA--')(A)
PUT SKIP
PUT DATA(PTLIB,PTOUT,ZONE1,I,J,AUXOP3,AUXOP4)
PUT SKIP EDIT('LIBRARY CONTENTS--')(A)
PUT SKIP EDIT (LIBRARY1(PTLIB))(A)
IF DUMPI.NO=1 THEN DO
PUT SKIP EDIT('REQUIRED DUMP--')(A)
CALL THEUMJ
  END
GOTO COMMONER
END
ON CONVERSION BEGIN
PUT SKIP EDIT('CONVERSION ERROR DETECTED DURING THE GENERATION
OF STATEMENT NUMBER',PTLIB-ADDR,'OF MACRO',NOM)
      (A,F(4),X(1),A,X(1),A)
PUT SKIP EDIT('CHECK THE VALIDITY OF THE ARGUMENTS')(A)
PUT SKIP EDIT('INDICATIVE DATA--')(A)
PUT SKIP
PUT DATA(PTLIB,PTOUT,ZONE1,I,J,AUXOP3,AUXOP4)
PUT SKIP EDIT('LIBRARY CONTENTS--')(A)
PUT SKIP EDIT (LIBRARY1(PTLIB))(A)
GOTO COMMONER
END
/*INITIALISATIONS*/
PTAUX=1
PTLOAR1,PTSCOPE,PTLOAR2=1
PTLOCHAR1,PTLOCHAR2=1
PTCONO=1
NIVD=1
PTSTACK=1
IF EXTRN=0 THEN DO
SWITCH1=SWITCH(PTDIR1)
SWITCH(PTDIR1)=1
  END
ELSE DO
I=INDEX(ACTMAC,NOM)
IF I =0 THEN SWITCH1=1
  ELSE DO
SUBSTR(ACTMAC,PTACT,6)=NOM
PTACT=PTACT+7
SWITCH1=0  END
  END
D10
CAT=0
ZONE1=' '
FIND=0
I=1 FIELD=1
J=7
AUXOP2=LABEL(PTLIB)
SEP=ANAL4(1)
IF SEP='-' THEN GOTO NOP
D60
IF SEP=' ' THEN GOTO DLAB1
IF SEP=' ' THEN GOTO D3
NCA=INDEX(AUXOP2,CATOP)
IF NCA=0 THEN DO
SUBSTR(ZONE1,I,J)=AUXOP2
GO TO D5  END
AUXOP4=SUBSTR(AUXOP2,1,NCA-1)
GO TO D7
DLAB10
IF INDEX(LETTER,ANAL4(2))=0 THEN GO TO DLAB2
NCA=3
DLAB30
SEP=ANAL4(NCA)
IF SEP=' ' THEN GO TO D4
IF SEP=CATOP THEN DO
CAT=1
GO TO D4  END
IF NCA=15 THEN GO TO DERR1

```

```

    NCA=NCA+1
    GO TO DLAB3
DLAB20
    IF ANAL4(4)=CATOP THEN CAT=1
    PNT=SUBSTR(AUXOP2,2,2)
    AUXOP4=ARGUM(PNT,NIVD)
    IF CAT=1 THEN DO
        NCA=4 GO TO D7 END
        SUBSTR(ZONE1,I,J)=AUXOP4
    GO TO D5
D40 INTERNAL=SUBSTR(AUXOP2,2,NCA-2)
    CALL SEARCHINT /*SEARCH INTERNAL VALUE*/
    IF FIELD=2 THEN GOTO DLAMT16
        OPCODE=CODE(PTLIB)
        INT=INDEX(STRING2,OPCODE)
        IF INT=0 THEN DO
            IF TYPE=0 THEN GOTO DERR2
            ELSE IF CAT=1 THEN DO
                AUXOP4=INTCHAR
                GOTO D7 END
            ELSE DO
                CAT=1
                NCA=0
                GOTO DLAMT12 END
            END
            IF INT=1 THEN
                IF TYPE=0 THEN GOTO SETA
                ELSE GOTO DERR12
                IF TYPE=0 THEN GOTO DERR13
                IF CAT=1 THEN GOTO DERR3
                ELSE GOTO SETC
DLAMT150
    K=15
DLAMT90
    IF ANALCHAR(K)=' ' THEN
        IF K,1 THEN DO
            K=K-1
            GOTO DLAMT9
        END
        SUBSTR(ZONE1,I,K)=INTCHAR
DLAMT50
    IF K)6 THEN GOTO D9
    I=1
    J=10+K+1
    CAT=0
    K=INDEX(''('',ANAL3(I))
    IF K=0 THEN GOTO D20
    FIELD=0
    IF K=1 THEN GOTO D10
    ELSE GOTO D10A
DLAMT160
    IF TYPE=0 THEN GOTO DLAMT7
    IF CAT=0 THEN GOTO DLAMT15
    AUXOP4=INTCHAR
    GOTO D7
DLAMT70
    CONVERT=INTARIT
    K=1
DLAMT80
    IF ANALCONV(K)=' ' THEN
        IF K)14 THEN DO
            K=K+1
            GOTO DLAMT8
        END
        IF CAT=1 THEN DO
            AUXOP4=SUBSTR(CONVERT,K)
            GOTO D7
        END
        SUBSTR(ZONE1,I,14-K+1)=SUBSTR(CONVERT,K)
        K=14-K+1
        GOTO DLAMT5
D70
    IF ANAL4(NCA+1)='£' THEN DO
        INTCHAR=SUBSTR(AUXOP2,NCA+1)

```

```

GO TO D8      END
IF INDEX(LETTER, ANAL4(NCA+2))=0 THEN DO
PNT=SUBSTR(AUXOP2, NCA+2, 2)
INTCHAR=ARGUM(PNT, NIVD)
GO TO D8      END
INTERNAL=SUBSTR(AUXOP2, NCA+2)
CALL SEARCHINT /*SEARCH INTERNAL VALUE*/
IF TYPE =0 THEN GO TO D8
CONVERT=INTARIT
K=1
DLAB40
IF ANALCONV(K) =' ' THEN GOTO DLAB5
IF K=14 THEN GOTO DLAB6
K=K+1
GO TO DLAB4
DLAB50
CONVERT=SUBSTR(CONVERT, K)
DLAB60
INTCHAR=CONVERT
D80
NCA=15
DLAMT100
IF ANAL5(NCA)=' ' THEN
IF NCA,1 THEN DO
NCA=NCA-1
GOTO DLAMT10
END
SUBSTR(ZONE1, I, NCA)=AUXOP4
DLAMT120
K=15
DLAMT110
IF ANALCHAR(K)=' ' THEN
IF K,1 THEN DO
K=K-1
GOTO DLAMT11
END
SUBSTR(ZONE1, I+NCA, K)=INTCHAR
K=NCA+K
IF FIELD=2 THEN GOTO DLAMT5
IF K)=7 THEN GOTO DLAMT6
ELSE GOTO DLAMT13
D50
IF FIELD=2 THEN GO TO D9
DLAMT60
IF SUBSTR(ZONE1, 1, 7) =' ' THEN
ANAL1(8)='='
D30
IF FIELD=2 THEN GO TO D9
DLAMT130
AUXOP2=CODE(PTLIB)
SEP=ANAL4(1)
AUXOP3=OPERAND(PTLIB)
OPCODE=AUXOP2
RECORD=AUXOP2
I=INDEX(STRING1, RECORD)
IF I =0 THEN DO
I= I/8+1
GO TO INTLABEL(I)
END
IF OPCODE='CAB ' THEN GOTO CAB
IF OPCODE='UAB ' THEN GOTO UAB
IF OPCODE='NOP ' THEN DO
NOP0
PTLIB=PTLIB+1
GOTO D1
END
IF OPCODE='NOTE' THEN GOTO NOTE
IF OPCODE='MEND' THEN GOTO MEND
IF OPCODE='NPUN' THEN DO
NOPUNCH=2
GOTO NOP
END
IF SEP='*' THEN GO TO NESTCALL
FIELD=2

```

```

IF CAT=1 THEN
IF K,7 THEN DO
I=K+1
GOTO DLAMT14
END
I=10
DLAMT140
J=4
GO TO D6
D90 /*OPERAND ANALYSIS*/
FIELD=0
CAT=0
I=1
J=16
DLAB70
SEP=ANAL3(I)
IF SEP='&' THEN GOTO DLAB8A
IF FIELD =0 THEN GO TO D10B
IF SEP='''' THEN GOTO D10
IF SEP='{' THEN GOTO D10A
D10B0
INT=2
DLAB80
NCA=1
DLAB90
IND=I+NCA
SEP=ANAL3(IND)
IF INDEX('+-',SEP) =0 THEN GOTO D11
IF SEP=CATOP THEN DO
CAT=1 GO TO D11 END
IF IND=50 THEN GOTO DERR1
NCA=NCA+1
GO TO DLAB9
DLAB8A0
I=I+1
IF INDEX(LETTER,ANAL3(I)) =0 THEN DO
INT=1 GO TO DLAB8 END
PNT=SUBSTR(AUXOP3,I,2)
RECORD=ARGUM(PNT,NIVD)
NCA=7
DLAB100
IF ANAL2(NCA) =' ' THEN GO TO DLAB11
IF NCA,1 THEN DO
NCA=NCA-1
GO TO DLAB10 END
DLAB110
SUBSTR(ZONE1,J,NCA)=RECORD
IF ANAL3(I+2)=CATOP THEN DO
J=J+NCA
I=I+3
GO TO D12
END
ANAL1(J+NCA)=ANAL3(I+2)
I=I+3
J=J+NCA+1
GO TO D13
D110
IF INT =2 THEN GO TO D14
SUBSTR(ZONE1,J,NCA+1)=SUBSTR(AUXOP3,I)
I=IND+1
J=J+NCA+1
D130IF CAT=1 THEN GO TO D18
D200
IF ANAL3(I)=' ' THEN
IF I=50 THEN GOTO DLAB12
ELSE DO
I=I+1
GO TO D20 END
IF INDEX(COMMENT,SUBSTR(AUXOP3,I,2)) =0 THEN DO
IF J)=26 THEN J=26
SUBSTR(ZONE1,J)=SUBSTR(AUXOP3,I)
GO TO DLAB12 END
SEP=ANAL3(I)
IF SEP='+' SEP='-' THEN DO

```

Z/MB.

```

ANAL1(J)=ANAL3(I)
J=J+1
I=I+1
GO TO D20          END
GO TO D12          Z/MB.
DLAB120
  ANAL1(70)='+'
  PUT FILE(SORTIE) EDIT(ZONE1) (A(80))
  PTOUT=PTOUT+1
  PTLI8=PTLI8+1
  GO TO D1
D120FIELD=1
  GO TO DLAB7
D140
  INTERNAL=SUBSTR(AUXOP3,I,NCA)
  CALL SEARCHINT
  IF TYPE=0 THEN GO TO D17
  NCA=15
DLAB130
  IF ANALCHAR(NCA)=' ' THEN GO TO DLAB14
  IF NCA=1 THEN GO TO DLAB14
  NCA=NCA-1
  GO TO DLAB13
DLAB140
  SUBSTR(ZONE1,J,NCA)=INTCHAR
D160J=J+NCA
  ANAL1(J)=SEP
  I=IND+1
  J=J+1
  GO TO D13
D170
  CONVERT=INTARIT
  NCA=1
DLAB150
  IF ANALCONV(NCA)=' ' THEN GO TO DLAB16
  IF NCA=14 THEN GO TO DLAB16
  NCA=NCA+1
  GO TO DLAB15
DLAB160
  SUBSTR(ZONE1,J,15-NCA)=SUBSTR(CONVERT,NCA)
  NCA=15-NCA
  GOTO D16
D180
  CAT=0
  J=J-1
  GO TO D12
D100
  SEP='''''
  IF ANAL3(I+1)=''''' THEN DO
    SUBSTR(ZONE1,J,2)='''''''
    I=I+2
    J=J+2          END
  ELSE DO
    ANAL1(J)='''''
DLAB200          Z/MB.
  I=I+1
  J=J+1  END
DD200
  IF ANAL3(I)='£' THEN GO TO DLAB21
  INT=2
DLAB230NCA=1
DLAB220
  IND=I+NCA
  SEP1=ANAL3(IND)
  IF SEP1=SEP THEN GO TO DD11
  IF SEP1=CATOP THEN GO TO DD16
  IF IND=50 THEN GO TO DERR1
  NCA=NCA+1
  GO TO DLAB22          Z/MB.
DLAB210
  I=I+1
  IF INDEX(LETTER,ANAL3(I))=0 THEN DO
    INT=1
    GO TO DLAB23  END

```

```

PNT=SUBSTR(AUXOP3,I,2)
INT=0
AUXOP2=ARGUM(PNT,NIVD)
IND=I+2
IF ANAL3(IND)=CATOP THEN CAT=1
NCA=7
GOTO DLAB24
D10A0
ANAL1(J)='('
SEP=')'
GO TO DLAB20
DD110
IF INT =2 THEN GO TO DD17
SUBSTR(ZONE1,J,NCA)=SUBSTR(AUXOP3,I)
DD130 ANAL1(J+NCA)=SEP
ANAL1(J+NCA+1)=ANAL3(IND+1) Z/MB.
I=IND+2
J=J+NCA+3
GOTO D20
DD170
INTERNAL=SUBSTR(AUXOP3,I,NCA) Z/MB.
CALL SEARCHINT
IF TYPE=0 THEN GO TO DD15
AUXOP2=INTCHAR
NCA=15
DLAB240 Z/MB.
IF ANAL4(NCA) =' ' THEN GO TO DLAB25
IF NCA,1 THEN DO
NCA=NCA-1
GO TO DLAB24 END
DLAB250
SUBSTR(ZONE1,J,NCA)=AUXOP2
IF CAT =0 THEN GO TO DD18
DLAB260
IF ANAL3(IND)=SEP THEN GO TO DD13
IF IND=50 THEN GOTO DERR1
IND=IND+1
GO TO DLAB26
DD150 Z/MB.
CONVERT=INTARIT
K=1
DLAB270 Z/MB.
IF ANALCONV(K) =' ' THEN GO TO DLAB28
IF K=14 THEN GO TO DLAB28
K=K+1
GO TO DLAB27
DLAB280 Z/MB.
NCA=15-K
SUBSTR(ZONE1,J,NCA)=SUBSTR(CONVERT,K) Z/MB.
IF CAT=0 THEN GO TO DD13
ELSE GO TO DD18
DD160 Z/MB.
CAT=1
IF INT=1 THEN GO TO DD17
SUBSTR(ZONE1,J,NCA)=SUBSTR(AUXOP3,I,NCA)
DD180 Z/MB.
J=J+NCA
IF FIELD=0 THEN DO
FIELD=1
I=IND+1
GOTO DD20 END
SUBSTR(ZONE1,J,21)=SUBSTR(AUXOP3,IND,2)
J=J+4
I=IND+2
GOTO D20
UAB0
PNT=SUBSTR(AUXOP3,42,9)
PTLIB=PTLIB+PNT
GO TO DI
CAB0
REVERT=0
IF ANAL3(2)='E' THEN GO TO DLAB29
J=INDEX(AUXOP3,' ')
AUXOP1=SUBSTR(AUXOP3,2,J-2) Z/MB.

```

```

INT=2
GO TO DLAB30
DLAB290
  IF INDEX(LETTER,ANAL3(3)) =0 THEN GO TO DLAB31
  PNT=SUBSTR(AUXOP3,3,2)
  AUXOP1=ARGUM(PNT,NIVD)
  INT=0
  J=5
  GO TO DLAB30
DLAB310
  J=INDEX(AUXOP3,' ')
  INTERNAL=SUBSTR(AUXOP3,3,J-3)
  CALL SEARCHINT
  INT=1
  IF TYPE=1 THEN AUXOP1=INTCHAR
  ELSE INTARIT1=INTARIT
DLAB300
  ORDRE=SUBSTR(AUXOP3,J+1,2)
I=INDEX(OPERA,ORDRE)
  I=I/3+1
  J=J+4
  IND=INDEX(AUXOP3,'')
  IF IND=0 THEN GO TO DERR1
D300
  IF ANAL3(J)='2' THEN GOTO DDLAB31
  IF ANAL3(J) =''' THEN GO TO D31
  AUXOP2=ANAL3(J+1)
  IND=3
  GO TO COMMON
DDLAB310
IF INDEX(LETTER,ANAL3(J+1)) =0 THEN GOTO D32
  PNT=SUBSTR(AUXOP3,J+1,2)
  AUXOP2=ARGUM(PNT,NIVD)
  IND=0
  GO TO COMMON
D310
  AUXOP2=SUBSTR(AUXOP3,J,IND-J)
  IND=2
  GO TO COMMON
D320
  INTERNAL=SUBSTR(AUXOP3,J+1,IND-J-1)
  TYPE1=TYPE
  CALL SEARCHINT
  IF TYPE=1 THEN AUXOP2=INTCHAR
  IND=1
COMMON0
  GOTO OPLAB(I)
EQ0
  IF INT=0 THEN
    IF IND=1 THEN GOTO BR13
    ELSE GOTO BR1
  IF INT=1 THEN GOTO AUXLAB1(IND)
  IF IND=1 THEN GOTO BR13 ~
  IF IND=0 THEN GOTO BR1
  GOTO BRER1
NE0
  REVERT=1
  GOTO EQ
  GTO
  IF INT=0 THEN DO
    IF IND=1 THEN GOTO BR17
    IF IND=3 THEN GOTO BRER2
  GOTO BR3   END
  IF INT=1 THEN DO
    IF IND=1 THEN GOTO BR18
    IF IND=3 THEN GOTO BRER2
  GOTO BR9   END
  GOTO AUXLAB3(IND)
GE0
  IF INT=0 THEN DO
    IF IND=1 THEN GOTO BR19
    IF IND=3 THEN GOTO BRER2
  GOTO BR4   END
  IF INT=1 THEN DO

```

```

        IF IND=1 THEN GOTO BR20
        IF IND=3 THEN GOTO BRER2
        GOTO BR10      END
        GOTO AUXLAB4(IND)
        LTO
        REVERT=1
        GOTO GE
LEO
        REVERT=1
        GOTO GT
NGO
        GOTO LE
NLO
        GOTO GE
BR10
        IF AUXOP1=AUXOP2 THEN DO
        IF REVERT=1 THEN GOTO NOP
        D34OPTLIB=PTLIB+SUBSTR(AUXOP3,42,9)
        END
        ELSE DO
        IF REVERT=1 THEN GOTO D34
        ELSE GOTO NOP
        END
        GO TO D1
BR30
        INTARIT1=AUXOP1
D350
        INTARIT2=AUXOP2
D430
        IF INTARIT1,INTARIT2 THEN
        IF REVERT=0 THEN GO TO D34
        ELSE GOTO NOP
        ELSE
        IF REVERT=0 THEN GOTO NOP
        ELSE GO TO D34
BR40
        INTARIT1=AUXOP1
D360
        INTARIT2=AUXOP2
D500
        IF INTARIT1,=INTARIT2 THEN
        IF REVERT=0 THEN GO TO D34
        ELSE GOTO NOP
        ELSE
        IF REVERT=0 THEN GOTO NOP
        ELSE GO TO D34
BR70
        IF TYPE =0 THEN GOTO BRI
D600
        INTARIT2=AUXOP2
D400
        IF INTARIT1=INTARIT2 THEN
        IF REVERT=1 THEN GOTO NOP
        ELSE GOTO D34
        ELSE
        IF REVERT=1 THEN GO TO D34
        ELSE GOTO NOP
BR90
        IF TYPE=0 THEN GO TO D35
        ELSE GO TO DERR4
BR100
        IF TYPE=0 THEN GO TO D36
        ELSE GO TO DERR4
BR130
        IF TYPE =0 THEN GOTO BRI
D410
        INTARIT1=AUXOP1
D420
        INTARIT2=INTARIT1
        GOTO D40
BR140
        IF TYPE=0 THEN
        IF TYPE1=0 THEN GOTO D42
        ELSE GO TO DERR4

```

Z/MB.

```
ELSE
  IF TYPE1=0 THEN GO TO DERR5
  ELSE GOTO BR1
BR170
  IF TYPE =0 THEN GO TO DERR4
  INTARIT1=AUXOP1
DLAB400
  INTARIT2=INTARIT
  GO TO D43
BR180
  IF TYPE=0 THEN
    IF TYPE1=0 THEN GO TO DLAB40
  GO TO DERR5
BR190
  IF TYPE =0 THEN GO TO DERR4
  INTARIT1=AUXOP1
DLAB410
  INTARIT2=INTARIT
  GO TO D50
BR200
  IF TYPE=0 THEN
    IF TYPE1=0 THEN GO TO DLAB41
  GO TO DERR5
BR250
  IF TYPE=0 THEN GO TO D60
  ELSE GOTO BR1
BR270
  IF TYPE=0 THEN GO TO DERR6
  ELSE GOTO BR1
BRER10
  GOTO DERR1
BRER20
  GOTO DERR7
DCLA0
  J=1
D1010
  FIN=0
D1000
  I=INDEX(AUXOP3,' ')
  IF I=0 THEN DO
DLAB50 OFIN=1
  I=INDEX(AUXOP3,' ') GO TO DLAB51 END
  IF ANAL3(I+1)='- ' THEN GO TO DLAB50
DLAB510 INTERNAL=SUBSTR(AUXOP3,1,I-1)
  FIND=1
  CALL SEARCHINT
  GO TO DCLLAB(J)
DCLLAB(1)0
  IF PTLOAR2,30 THEN DO
    PRINT00 =PRINT13
  PERR10
    PUT SKIP EDIT(PRINT9,PRINT18,PRINT00,PRINT23,PRINT1,PTLIB-ADDR,
PRINT7,NOM,WARN4)(R(F4))
F40FORMAT(A,X(1),A,X(1),A,X(1),A,X(1),A,F(4),X(1),A,X(1),A,X(1),A)
    GOTO COMMONER END
    IDLOAR(PTLOAR2)=INTERNAL
    VALOAR(PTLOAR2)=0
    PTLOAR2=PTLOAR2+1
  D1030
    IF FIN=0 THEN DO
      AUXOP3=SUBSTR(AUXOP3,I+1)
      GO TO D100 END
    GOTO NOP
DCLLAB(3)0
  IF PTLOCHAR2,30 THEN DO
    PRINT00=PRINT12
    GO TO PERR1 END
    IDLOCHAR(PTLOCHAR2)=INTERNAL
    VALOCHAR(PTLOCHAR2)=' '
    PTLOCHAR2=PTLOCHAR2+1
    GO TO D103
DCLLAB(2)0
  IF PTGBAR,110 THEN DO
    PRINT00=PRINT15
```

```

GO TO PERRI      END
IDGCHAR(PTGBAR)=INTERNAL
PTGBAR=PTGBAR+1
GO TO D103
DCLLAB(4)0
IF PTGBCH,100 THEN DO
  PRINT00=PRINT14
  GO TO PERRI      END
IDGCHAR(PTGBCH)=INTERNAL
PTGBCH=PTGBCH+1
GO TO D103
DCLC0
  J=3
  GO TO D101
DCGA0
  IF SWITCH1=0 THEN GOTO NOP
  J=2
  GO TO D101
DCGC0
  IF SWITCH1=0 THEN GOTO NOP
  J=4
  GO TO D101
SETA0
  PTAUX=PID
  TYPEAUX=LOCAL
  IF LOCAL,2 THEN GO TO DERR12
  AUXOP3=OPERAND(PTLIB)
  I=1
  IF ANAL3(I)='&' THEN DO
    INT=1
    I=I+1
  IF INDEX(LETTER,ANAL3(I))=0 THEN GOTO DDLAB50
  END
  ELSE INT=2
  NCA=I
DDLAB510
  IND=I+NCA
  SEP=ANAL3(IND)
  J=INDEX(' -+*/',SEP)
  IF J=0 THEN DO
    IF IND=50 THEN GO TO DERR1
    NCA=NCA+1
    GO TO DDLAB51  END
  INTERNAL=SUBSTR(AUXOP3,I,NCA)
  I=IND
D2000
  I=I+1
  GO TO D201
DDLAB500
  PNT=SUBSTR(AUXOP3,I,2)
  INT=0
  INTERNAL=ARGUM(PNT,NIVD)
  I=I+2
  J=INDEX(' -+*/',ANAL3(I))
  IF J=0 THEN GO TO DERR1
  ELSE GO TO D200
D2010
  IF INT=1 THEN DO
    CALL SEARCHINT
    IF TYPE=0 THEN INTARIT1=INTARIT
    ELSE INTARIT1=INTCHAR
    END
  ELSE INTARIT1=INTERNAL
  IF J=1 THEN GO TO D203
  IF ANAL3(I)='&' THEN GO TO DLAB52
  INT=2
DLAB530
INTERNAL=SUBSTR(AUXOP3,I,INDEX(SUBSTR(AUXOP3,I,2),I-1)-1)
GO TO D202
D2030
  IF TYPEAUX=1 THEN VALOAR(PTAUX)=INTARIT1
  ELSE VAGDAR(PTAUX)=INTARIT1
  GOTO NOP
DLAB520

```

Z/MB.

Z/MB.

```
I=I+1
IF INDEX(LETTER, ANAL3(I)) =0 THEN DO
INT=1
GO TO DLAB53      END
PNT=SUBSTR(AUXOP3,I,2)
INTARIT2=ARGUM(PNT,NIVD)
DLAB540
GO TO LABARIT(J)
D2020
IF INT =1 THEN DO
INTARIT2=INTERNAL
GO TO DLAB54      END
CALL SEARCHINT
IF TYPE=0 THEN INTARIT2=INTARIT
ELSE INTARIT2=INTCHAR
GO TO DLAB54
LABARIT(2)0
INTARIT1=INTARIT1-INTARIT2
GO TO D203
LABARIT(3)0
INTARIT1=INTARIT1+INTARIT2
GO TO D203
LABARIT(4)0
INTARIT1=INTARIT1*INTARIT2
GO TO D203
LABARIT(5)0
INTARIT1=INTARIT1/INTARIT2
GO TO D203
SETCO
CAT=0
PTAUX=PID
TYPEAUX=LOCAL
IF LOCAL=2 THEN GO TO DERR13
AUXOP3=OPERAND(PTLIB)
I=1
IF ANAL3(I)='&' THEN DO
I=I+1
IF INDEX(LETTER, ANAL3(I))=0 THEN GOTO DLAB60
INT=1      END      ELSE DO
IF ANAL3(I)='' THEN GO TO D302
IF SUBSTR(AUXOP3,1,4)='MOVE' THEN GO TO D306
INT=2      END
NCA=1
DLAB610
IND=I+NCA
SEP=ANAL3(IND)
IF SEP=' ' THEN GO TO D300
IF SEP=CATOP THEN DO
CAT=1      GO TO D300      END
IF IND=50 THEN GO TO DERR1
NCA=NCA+1
GOTO DLAB61
DLAB600
PNT=SUBSTR(AUXOP3,I,2)
INTCHAR=ARGUM(PNT,NIVD)
I=I+2
IF ANAL3(I)=CATOP THEN GO TO D303
ELSE GO TO D305
D3000
INTERNAL=SUBSTR(AUXOP3,I,NCA)
I=IND
IF INT =1 THEN DO
INTCHAR=INTERNAL
GO TO D301
END
CALL SEARCHINT
IF TYPE =0 THEN GO TO D301
CONVERT=INTARIT
K=1
DLAB620
IF ANALCONV(K)=' ' THEN
IF K=14 THEN GOTO DLAB63
ELSE DO
K=K+1
```

Z/MB.
Z/MB.

```

GO TO DLAB62  END
DLAB630
  INTCHAR=SUBSTR(CONVERT,K)
D3010
  IF CAT=1 THEN GO TO D303
D3050
  IF TYPEAUX=3 THEN VALOCHAR(PTAUX)=INTCHAR
  ELSE VAGCHAR(PTAUX)=INTCHAR
  GOTO NOP
D3020
  INTCHAR=ANAL3(I+1)
  I=I+3
  IF ANAL3(I)=CATOP THEN GOTO D303
  ELSE GO TO D305
D3030
  I=I+1
  AUXOP4=INTCHAR
  IF ANAL3(I)='£' THEN DO
    I=I+1
    IF INDEX(LETTER,ANAL3(2))=0 THEN GOTO DLAB64
    INT=1
    END
  ELSE DO
    IF ANAL3(I)='''' THEN DO
      INTCHAR=ANAL3(I+1)
GO TO D304  END
    INT=2 END
    INTERNAL=SUBSTR(AUXOP3,I,INDEX(SUBSTR(AUXOP3,I),' ')-1)
    IF INT =1 THEN DO
      INTCHAR=INTERNAL
      GOTO D304  END
    CALL SEARCHINT
    IF TYPE =0 THEN GO TO D304
    CONVERT=INTARIT
    K=1
DLAB650 IF ANALCONV(K)=' ' THEN DO
  IF K=14 THEN GOTO DLAB66
  K=K+1
  GOTO DLAB65  END
DLAB660
  INTCHAR=SUBSTR(CONVERT,K)
D3040
  K=15
DLAB670
  IF ANAL5(K)=' ' THEN DO
    IF K=1 THEN GO TO DLAB68
    K=K-1
    GOTO DLAB67  END
DLAB680 INTCHAR=SUBSTR(AUXOP4,1,K)  INTCHAR
  GOTO D305
DLAB640
  PNT=SUBSTR(AUXOP3,I,2)
  INTCHAR=ARGUM(PNT,NIVD)
  GOTO D304
D3060
  FIN=0
  I=6
IND=INDEX(AUXOP3,',')
  IF IND=0 THEN GOTO DERR1
  IF ANAL3(I)='£' THEN DO
    INTCHAR=SUBSTR(AUXOP3,I,IND-I)
    GO TO DLAB70  END
    I=I+1
    IF INDEX(LETTER,ANAL3(I))=0 THEN DO
      PNT=SUBSTR(AUXOP3,I,2)
      INTCHAR=ARGUM(PNT,NIVD)
      GO TO DLAB70  END
    INTERNAL=SUBSTR(AUXOP3,I,IND-I)
    CALL SEARCHINT
    IF TYPE=0 THEN GOTO DERR14
DLAB700
  I=IND+1
  K=1
  SEP=', '
D3080

```

Z/MB.

Z/MB.

Z/MB.

```

AUXOP3=SUBSTR(AUXOP3,1)
IND=INDEX(AUXOP3,SEP)
IF IND=0 THEN GOTO DERR1
IF ANAL3(1)='&' THEN DO
MOVEUP(K)=SUBSTR(AUXOP3,1,IND-1)
GO TO D307      END
IF INDEX(LETTER,ANAL3(2))=0 THEN DO
PNT=SUBSTR(AUXOP3,2,2)
MOVEOP(K)=ARGUM(PNT,NIVD)      END
ELSE DO
INTERNAL=SUBSTR(AUXOP3,2,IND-2)
CALL SEARCHINT
IF TYPE=0 THEN GOTO DERR15
MOVEOP(K)=INTARIT      END
D3070
IF FIN=1 THEN GOTO DLAB71
K=K+1
IF K,2 THEN DO
SEP=',)'
FIN=1      END
I=IND+1
GO TO D308
DLAB710
IF TYPEAUX=3 THEN
SUBSTR(VALUCHAR(PTAUX),MOVEOP(3))=SUBSTR(INTCHAR,MOVEOP(1),MOVE
LP(2))
ELSE
SUBSTR(VAGCHAR(PTAUX),MOVEOP(3))=SUBSTR(INTCHAR,MOVEOP(1),MOVEOP(2))
GOTO NUP
NESTCALLO
INT=0
NIVD=NIVD+1
FIN=0
IF NIVD,10 THEN GO TO DERR16
NESTERR=1
ARGUM(*,NIVD)=' '
NESTMATRIX=0
CONDSTACK(PTCOND)=SWITCH1
CONDSTACK(PTCOND+1)=EXTRN
PTCOND=PTCOND+2
NESTNAME(PTSTACK)=NOM
NESTADDR(PTSTACK)=ADDR
NOM1=SUBSTR(CODE(PTLIB),2,6)
PNT=1
DLAB800
IF NAME(PNT)=NOM1 THEN GOTO NEST1
IF PNT=PTDIR THEN GOTO NEST10
PNT=PNT+1
GO TO DLAB80
NEST10
SWITCH1=SWITCH(PNT)
SWITCH(PNT)=1
EXTRN=0
/PTAUX=ADRESSE(PNT)
NEST110
K=0
AUXOP4=LABEL(PTLIB)
IF ANAL5(1)='&' THEN DO
IF INDEX(LETTER,ANAL5(2))=0 THEN DO
INTERNAL=SUBSTR(AUXOP4,2,6)
NESTLAB=1
GOTO NEST17      END
PNT=SUBSTR(AUXOP4,2,2)
NESTMATRIX(K)=PNT      END
ELSE DO
ARGUM(K,NIVD)=AUXOP4
NEST180
NESTMATRIX(K)=999      END
AUXOP3=OPERAND(PTLIB)
NEST50
I=1
NEST40
IF ANAL3(I)='&' THEN GO TO DLAB81
NEST120

```

Z/MB.

Z/MB.

```
NCA=1
DLAB820
  IND=I+NCA
  IF IND=50 THEN GOTO NEST6
  IF ANAL3(IND)=',' THEN GO TO NEST3
  NCA=NCA+1
  GOTO DLAB82
DLAB810
  I=I+1
  IF INDEX(LETTER,ANAL3(I)) =0 THEN DO
    INT=1
    GOTO NEST12    END
  PNT=SUBSTR(AUXOP3,I,2)
  K=K+1
  NESTMATRIX(K)=PNT
  I=I+2
  IF I,50 THEN GO TO NEST7
    ELSE GO TO NEST2
NEST30
  K=K+1
  IF INT=1 THEN GOTO NEST15
  ARGUM(K,NIVD)=SUBSTR(AUXOP3,I,NCA)
NEST160
  NESTMATRIX(K)=999
  IF FIN =0 THEN GOTO DLAB83
  I=IND
NEST20
  I=I+1
  IF ANAL3(I)='/' THEN GOTO NEST8
  IF ANAL3(I)=',' THEN GO TO NEST7
  GO TO NEST4
DLAB830
  FIN=0
NEST70
  IF SUBSTR(CODE(PTLIB),15,1)='/' THEN GOTO NEST8
  PTLIB=PTLIB+1
  AUXOP3=OPERAND(PTLIB)
  GO TO NEST5
NEST60
  NCA=NCA+1
  FIN=1
  GO TO NEST3
NEST80
  PNT=0
DLAB840
  J=NESTMATRIX(PNT)
  IF J =999 THEN
    ARGUM(PNT,NIVD)=ARGUM(J,NIVD-1)
    IF PNT=K THEN GO TO NEST9
    PNT=PNT+1
    GO TO DLAB84
NEST90
  NESTSCOPE(PTSCOPE)=PTLOAR1
  NESTSCOPE(PTSCOPE+1)=PTLOAR2
  NESTSCOPE(PTSCOPE+2)=PTLOCHAR1
  NESTSCOPE(PTSCOPE+3)=PTLOCHAR2
  PTLOCHAR1=PTLOCHAR2
  PTLOAR1=PTLOAR2
  PTSCOPE=PTSCOPE+4
  ADDR=PTAUX-1
  NOM=NOM1
  NESTSTACK(PTSTACK)=PTLIB+1
  PTLIB=PTAUX
  PTSTACK=PTSTACK+1
  NESTERR=0
  GO TO D1
NEST100
  PTREC=DYNCORE(PTDYN-1)
  PTAUX=PTREC
  CALL FETCH(LIBRARY1,NOM1,PTREC,RESULT,END1)
  IF RESULT=0 THEN DO
    DYNCORE(PTDYN)=PTREC
    PTDYN=PTDYN+1
    EXTRN=1
```

```

I=INDEX(ACTMAC,NOM1)
IF I=0 THEN DO
  SWITCH1=0
  SUBSTR(ACTMAC,PTACT,6)=NOM1
  PTACT=PTACT+7
  END
ELSE SWITCH1=1
GOTO NEST11

      END
IF RESULT=4 THEN GO TO DERR17
PUT SKIP EDIT('I/O ERROR DETECTED WHEN RETRIEVING MACRO',NOM)
      (A,X(1),A)
GOTO COMMONER
NEST150
  INTERNAL=SUBSTR(AUXOP3,I,NCA)
  INT=0
NEST170
  CALL SEARCHINT
  IF TYPE=0 THEN DO
    CONVERT=INTARIT
    NCA=1
DLAB850
  IF ANALCONV(NCA)=0 THEN
    IF NCA)14 THEN DO
      NCA=NCA+1
      GOTO DLAB850      END
    INTCHAR=SUBSTR(CONVERT,NCA)
    END
    ARGUM(K,NIVD)=INTCHAR
    IF NESTLAB=1 THEN DO
      NESTLAB=0
      GOTO NEST18      END
    GOTO NEST16
MEND0
  IF EXTRN =0 THEN PTDYN=PTDYN-1
  IF NIVD)=1 THEN DO
    PTLOAR2=PTLOAR1
    PTLOCHAR2=PTLOCHAR1
DLAMT20
  RETURN      END
  PTLOCHAR2=NESTSCOPE(PTSCOPE-1)
  PTLOCHAR1=NESTSCOPE(PTSCOPE-2)
  PTLOAR2=NESTSCOPE(PTSCOPE-3)
  PTLOAR1=NESTSCOPE(PTSCOPE-4)
  PTSCOPE=PTSCOPE-4
  SWITCH1=CONDSTACK(PTCOND-2)
  EXTRN=CONDSTACK(PTCOND-1)
  PTCOND=PTCOND-2
  NOM=NESTNAME(PTSTACK-1)
  ADDR=NESTADDR(PTSTACK-1)
  PTLIB=NESTSTACK(PTSTACK-1)
  PTSTACK=PTSTACK-1
  NIVD=NIVD-1
  GO TO D1
NOTE0
  PUT SKIP LIST('***WARNING***')
  PUT EDIT(AUXOP3)(A(50))
  GOTO NOP
SEARCHINT0
  PROCEDURE /*THIS PROC DELIVERS THE ADDRESS AND THE VALUE OF AN
            INTERNAL VARIABLE IT ALSO DETECTS DOUBLE DECLARA-
            TIONS*/
    IF PTLOAR1=PTLOAR2 THEN GOTO SEARL1
    PID=PTLOAR1
SEARL30IF IDLOAR(PID)=INTERNAL THEN GOTO SEARL2
    PID=PID+1
    IF PID)PTLOAR2 THEN GOTO SEARL3
SEARL10
    IF PTGBAR=1 THEN GO TO SEARL4
    PID=1
SEARL70
    IF IDGOAR(PID)=INTERNAL THEN GOTO SEARL6
    PID=PID+1
    IF PID)PTGBAR THEN GO TO SEARL7

```

Z/MB.

Z/MB.

```

GO TO SEARL4
SEARL20
  IF FIND =1 THEN DO
    INTARIT=VALOAR(PID)
    LOCAL=1
    GO TO SEAR2      END
  IF J=1 THEN GOTO SEARWARN1
    ELSE GOTO SEARERR1
SEARL60
  IF FIND =1 THEN DO
    INTARIT=VAGOAR(PID)
    LOCAL=2
  SEAR20
    TYPE=0
    RETURN      END
  IF J=2 THEN GOTO D103
  ELSE GOTO SEARERR1
SEARL40
  IF PTLOCHAR1=PTLOCHAR2 THEN GOTO SEAR1
  PID=PTLOCHAR1
SEARL90IF IDLOCHAR(PID)=INTERNAL THEN GOTO SEARL8
  PID=PID+1
  IF PID)PTLOCHAR2 THEN GOTO SEARL9
SEARL10
  IF PTGBCH=1 THEN GOTO SEARL5
  PID=1
SEARL100
  IF IDGCHAR(PID)=INTERNAL THEN GOTO SEARL11
  PID=PID+1
  IF PID)PTGBCH THEN GOTO SEARL10
SEARL50
  IF FIND=1 THEN DO
    FIND=0
    RETURN      END
  PUT SKIP EDIT(PRINT19,INTERNAL,PRINT1,PTLIB-ADDR,PRINT7,NOM,ERR
OR42,PRINT20)
    (A,X(2),A,X(2),A,F(4),X(2),A,X(2),A,X(2),A)
    GOTO COMMONER
SEARL80
  IF FIND =1 THEN DO
    INTCHAR=VALOCHAR(PID)
    LOCAL=3
    GO TO SEAR3      END
  IF J=3 THEN GOTO SEARWARN1
    ELSE GOTO SEARERR1
SEARL110
  IF FIND =1 THEN DO
    INTCHAR=VAGCHAR(PID)
    LOCAL=4
  SEAR30
    TYPE=1
    RETURN      END
  IF J=4 THEN GOTO D103
    ELSE GO TO SEARERR1
  SEARWARN10
    PUT SKIP EDIT(PRINT21,PRINT19,INTERNAL,PRINT1,PTLIB-ADDR,PRINT7
,NOM,WARN4)
    (A,X(2),A,X(2),A,X(2),A,F(4),X(2),A,X(2),A,X(2),A)
    FIND=0
    GO TO D103
  SEARERR10
    PUT SKIP EDIT(PRINT22,INTERNAL,PRINT1,PTLIB-ADDR,PRINT7,NOM,PRI
NT20)
    (A,X(2),A,X(2),A,F(4),X(2),A,X(2),A,X(2),A)
    GOTO COMMONER
  END
DERR10 PUT SKIP EDIT (ERROR43) (A)
  CM10 PUT EDIT(PRINT1,PTLIB-ADDR,PRINT7,NOM,WARN4) (R(F1))
F10 FORMAT (X(2),A,F(5),X(2),A,X(2),A,X(2),A)
  GO TO COMMONER
DERR20
  PUT SKIP EDIT(ERROR44)(A) GO TO CM1
DERR30
  PUT SKIP EDIT(ERROR45) (A) GO TO CM1

```

```
DERR40
PUT SKIP EDIT(ERROR46)(A) GO TO CM1
DERR50
PUT SKIP EDIT(ERROR47)(A) GO TO CM1
DERR60
PUT SKIP EDIT(ERROR47,PRINT8) (A(46),X(2),A) GO TO CM1
DERR70
PUT SKIP EDIT(ERROR48)(A) GO TO CM1
DERR120
PUT SKIP EDIT(ERROR49) (A) GO TO CM1
DERR130
PUT SKIP EDIT(ERROR50) (A) GO TO CM1
DERR140
PUT SKIP EDIT(ERROR51) (A) GO TO CM1
DERR150
PUT SKIP EDIT(ERROR52) (A) GO TO CM1
DERR160
PUT SKIP EDIT(PRINT9,PRINT16) (A,X(2),A) GO TO CM1
DERR170
PUT SKIP EDIT(ERROR53)(A) GO TO CM1
COMMONERO
  IF NESTERR=1 THEN DO
    NIVD=NIVD-1
    PTCOND=PTCOND-2
    EXTRN=CONDSTACK(PTCOND+1)
    SWITCH1=CONDSTACK(PTCOND)
    NESTERR=0      END
    NOPUNCH=2
    GOTO NOP
  END
```

```
PRINTIT0
  PROCEDURE(SOPARM,CREPARM,LISTPARM,IDENPARM)
  DCL IDENPARM FIXED BINARY
  DCL SOPARM FIXED BINARY
  DCL CREPARM FIXED BINARY(31)
  DCL DUMPIND FIXED BINARY INIT(0) EXTERNAL
  DCL NOPUNCH FIXED BINARY INIT(0) EXTERNAL
  DCL LISTPARM FIXED BINARY
  DCL OPT CHAR(8)
  DCL TITLE1 CHAR(51) INIT('THE OPTIONS SPECIFIED FOR THE MACRO G
GENERATOR ARE--')
  DCL ASTERIX CHAR(46) INIT((46) '*')
  DCL TITLE CHAR(22) INIT('TAS-86 MACRO-GENERATOR')
  PUT PAGE
  PUT EDIT(ASTERIX,TITLE,ASTERIX)(A,X(2),A,X(2),A)
  PUT SKIP(3)
  PUT EDIT(TITLE1)(A)
  IF SOPARM=0 THEN
    OPT='SOURCE'
  ELSE
    OPT='NOSOURCE'
  PUT SKIP EDIT(OPT)(X(60),A)
  IF CREPARM=0 THEN
    OPT='GENERATE'
  ELSE
    IF CREPARM=1 THEN
      OPT='CREATE'
    ELSE
      IF CREPARM=2 THEN OPT='UPDATE'
    ELSE
      IF CREPARM=3 THEN OPT='DELETE'
    ELSE OPT='CHANGE'
  PUT SKIP EDIT(OPT)(X(60),A)
  IF NOPUNCH=0 THEN
    OPT='DECK'
  ELSE OPT='NODECK'
  PUT SKIP EDIT(OPT)(X(60),A)
  IF LISTPARM=0 THEN
    OPT='LIST'
  ELSE OPT='NOLIST'
  PUT SKIP EDIT(OPT)(X(60),A)
  IF IDENPARM=0 THEN OPT='NO-IDENT'
  ELSE OPT='IDENT'
  PUT SKIP EDIT(OPT)(X(60),A)
  IF DUMPIND=1 THEN
    PUT SKIP EDIT('DUMP')(X(60),A)
  END
```

PRINTERO

```

PROCEDURE(SOPARM,CREPARM,LISTPARM,IDENPARM)
DCL (SOPARM,LISTPARM) FIXED BINARY
DCL CREPARM FIXED BINARY(31)
DCL TITLE1 CHAR(24) INIT('TAS-86 GENERATED PROGRAM')
DCL ASTERIX CHAR(46) INIT((46) '*')
DCL NOPUNCH FIXED BINARY INIT(0) EXTERNAL
DCL IDENPARM FIXED BINARY
DCL IDENFIELD CHAR(3) INIT(' ')
DCL PTOUT FIXED BINARY(31) INIT(1) EXTERNAL
DCL SYSPNCH FILE STREAM OUTPUT
DCL SORTIE FILE STREAM
DCL ZONE1 CHAR(80)
DCL I FIXED BINARY INIT(1)
PUT PAGE
OPEN FILE(SORTIE) INPUT
GET FILE(SORTIE) EDIT(ZONE1)(A(80))
IF IDENPARM =0 THEN DO
1 J=INDEX(ZONE1,'NM')
1 IF J=0 THEN
1 PUT SKIP EDIT('INVALID NM INSTRUCTION.IDENT IGNORED.')(A)
1 ELSE
1 IDENFIELD=SUBSTR(ZONE1,J+3)
1 END
IF LISTPARM =0 THEN GOTO PR4
IF NOPUNCH =0 THEN GOTO PR8
PUT EDIT(ASTERIX,TITLE1,ASTERIX)(A,X(2),A,X(2),A)
PUT SKIP EDIT(I,ZONE1,IDENFIELD)
(F(4),X(3),A(72),X(3),A(3))
PUT FILE(SYSPNCH) EDIT(ZONE1,IDENFIELD,I)
(A(72),A(3),X(1),F(4))
PR50
I=I+1
IF I,PTOUT THEN GOTO PR1
GET FILE(SORTIE) EDIT(ZONE1)(A(80))
PUT SKIP EDIT(I,ZONE1,IDENFIELD)
(F(4),X(3),A(72),X(3),A(3))
PUT FILE(SYSPNCH) EDIT(ZONE1,IDENFIELD,I)
(A(72),A(3),X(1),F(4))
GOTO PR5
PR10
PUT SKIP(4) EDIT(PTOUT,'TAS-86 INSTRUCTIONS HAVE BEEN GENERATED')
(F(4),X(2),A)
RETURN
PR40
IF NOPUNCH =0 THEN GOTO PR2
PUT FILE(SYSPNCH) EDIT(ZONE1,IDENFIELD,I)
(A(72),A(3),X(1),F(4))
PR60
I=I+1
IF I,PTOUT THEN GOTO PR1
GET FILE(SORTIE) EDIT(ZONE1)(A(80))
PUT FILE(SYSPNCH) EDIT(ZONE1,IDENFIELD,I)
(A(72),A(3),X(1),F(4))
GO TO PR6
PR70
PUT SKIP EDIT('THE OUTPUT FILE SYSPNCH(TAPE OR PUNCHED CARDS) ,
HAS NOT BEEN WRITTEN DUE TO THE DETECTION OF SEVERE ERRORS DURING THE G
ENERATION.')(A)
GOTO PR1
PR80
PUT SKIP EDIT(ASTERIX,TITLE1,ASTERIX)(A,X(2),A,X(2),A)
PUT SKIP EDIT(I,ZONE1,IDENFIELD)
(F(4),X(3),A(72),X(3),A(3))
PR90
I=I+1
IF I,PTOUT THEN DO
1 PR20
1 IF NOPUNCH=1 THEN GOTO PR1
1 ELSE GOTO PR7
1 END
1 GET FILE(SORTIE) EDIT(ZONE1)(A(80))
1 PUT SKIP EDIT(I,ZONE1,IDENFIELD)
1 (F(4),X(3),A(72),X(3),A(3))
1 GOTO PR9
END

```

ECT CODE	ADDR1	ADDR2	STMT	SOURCE	STATEMENT
			1	FETCH	CSECT
			2		USING *,15
0	F00C	0000C	3		BC 15,SAVE
			4		DC AL1(7)
5E3C3C84040			5		DC C'FETCH '
C D00C	0000C		6	SAVE	STM 14,12,12(13) SAVE REGISTERS
			7		DROP 15
0			8		BALR 10,0
			9		USING *,10
0	A156	00168	10		ST 13,SAVEAREA+4 PROVIDE ADDRESSABII
D			11		LR 11,13 SET SAVE AREAS CONI
0	A152	00164	12		LA 13,SAVEAREA
B	0008	00008	13		ST 13,8(11)
1			14		LR 9,1 PARAMETER LIST ADDRESS
0	A024	00036	15	PASS1	BC 0,INSTR
			16		OPEN (MAC1DCB)
0			17+		CNOP 0,4
0	A01E	00030	18+		BAL 1,*,+8 LOAD REG1 W/LIST ADDR.
			19+		DC AL1(128) OPTION BYTE
0F4			20+		DC AL3(MAC1DCB) DCB ADDRESS
3			21+		SVC 19 ISSUE OPEN SVC
0	A011	00023	22		MVI PASS1+1,X'FO' OPEN DATA SET IN T
0	0010	00010	23	INSTR	L 3,16(9) LOAD END INDICATOR ADD
3	0000	00000	24		LH 3,0(3) LOAD END INDICATOR
3			25		LTR 3,3
0	A0D2	000E4	26		BNZ CLOSE
0	0004	00004	27		L 3,4(9) LOAD MACRO NAME DOF
3	0000	00000	28		L 3,0(3) LOAD MACRO'S NAME /
5	A146	3000 00158 00000	29		MVC NAME(6),0(3) MOVE MACRO NAME TO
			30	* SEARCH	FOR THE MACRO IN THE DIRECTORY
			31	FIND	MAC1DCB,NAME,D
0	A0E2	000F4	32+		LA 1,MAC1DCB LOAD PARAMETER REG 1
0	A146	00158	33+		LA 0,NAME LOAD PARAMETER REG 0
1			34+		LCR 1,1 INDICATE TYPE D
2			35+		SVC 18 ISSUE FIND SVC
0	000C	0000C	36	RESULT	L 3,12(9) LOAD RESULT ADDRESS
3	0000	00000	37		STH 15,0(3) STORE RETURN CODE IN RES
=			38		LTR 15,15 TEST RETURN CODE
0	A0C6	000D8	39		BNZ RETURN
0	0000	00000	40		L 3,0(9) LOAD LIBRARY DOPE V
3	0000	00000	41		L 5,0(3) LOAD LIBRARY VIRTUA
3	0004	00004	42		L 7,4(3) LOAD LIBRARY MULTIF
0	0008	00008	43		L 8,8(9) LOAD LIBRARY POINTE
3	0000	00000	44		LH 8,0(8) LOAD LIBRARY POINTER
7			45		LR 3,7 ADDRESS CALCULATION
3			46		MR 2,8 MULTIPLIER*POINTER
3			47		AR 5,3 VIRTUAL ORIGIN+ DIS
			48	*READ A RECORD	
			49	LOOP	READ DECB3,SF,MAC1DCB,(5)
0			50+		CNOP 0,4
0	A08E	000A0	51+LOOP		BAL 1,*,+24 LOAD DECB ADDRESS
00000			52+DECB3		DC F'0' EVENT CONTROL BLOCK
			53+		DC X'00' TYPE FIELD
			54+		DC X'80' TYPE FIELD
0			55+		DC AL2(0) LENGTH

ECT CODE	ADDR1	ADDR2	STMT	SOURCE	STATEMENT
000F4			56+	DC	A(MAC1DCB) DCB ADDRESS
00000			57+	DC	A(0) AREA ADDRESS
00000			58+	DC	A(0) RECORD POINTER WORD
1 000C		0000C	59+	ST	5,12(1,0) STORE AREA ADDRESS
1 0008		00008	60+	L	15,8(1,0) LOAD DCB ADDRESS
0 F030		00030	61+	L	15,48(0,15) LOAD RDWR ROUTINE ADDR
F			62+	BALR	14,15 LINK TO RDWR ROUTINE
			63	CHECK	DECB3
0 A07A		0008C	64+	LA	1,DECB3 LOAD PARAMETER REG 1
0 1008		00008	65+	L	14,8(0,1) PICK UP DCB ADDRESS
0 E034		00034	66+	L	15,52(0,14) LOAD CHECK ROUT. ADDR.
F			67+	BALR	14,15 LINK TO CHECK ROUTINE
0 A19E		00180	68	A	8,=F'1' INCREASE LIBRARY PI
3 500F A14E 0000F		00160	69	CLC	15(4,5),MEND TEST FOR FINAL RECI
0 A0BE		000D0	70	BE	FINAL
7			71	AR	5,7 UPDATE LIBRARY ADDI
0 A076		00088	72	BC	15,LOOP
9 0008		00008	73 FINAL	L	4,8(9) LOAD LIBRARY POINTER
4 0000		00000	74	STH	8,0(4) STORE NEW VALUE FOR POINTI
0 A156		00168	75 RETURN	L	13,SAVEAREA+4
C D00C		0000C	76	LM	14,12,12(13) RESTORE REGISTERS
E			77	BR	14
			78 CLOSE	CLOSE	(MAC1DCB)
0			79+	CNOP	0,4
0 A0DA		000EC	80+CLOSE	BAL	1,*+8 BRANCH AROUND LIST
			81+	DC	AL1(128) OPTION BYTE
0F4			82+	DC	AL3(MAC1DCB) DCB ADDRESS
4			83+	SVC	20 ISSUE CLOSE SVC
0 A0C6		000D8	84	BC	15,RETURN
			85 MAC1DCB	DCB	DSORG=PO,DUNAME=DDMAC,EODAD=ERR1,MACRI
			87+*		DATA CONTROL BLOCK
			88+*		
			89+	ORG	*-0 TO ELIMINATE UNUSED SPACE
			90+MAC1DCB	DS	OF ORIGIN ON WORD BOUNDARY
			91+	ORG	*+0 TO ORIGIN GENERATION
			93+*		DIRECT ACCESS DEVICE INTERFAC
00000000000000			95+	DC	BL16'0' FDAD,DVTBL
00000			96+	DC	A(0) KEYLE,DEVT,TRBAL
			98+*		COMMON ACCESS METHOD INTERFAC
			100+	DC	AL1(0) BUFNO
001			101+	DC	AL3(1) BUFCB
0			102+	DC	AL2(0) BUFL
0			103+	DC	BL2'0000001000000000' DSORG
00001			104+	DC	A(1) IOBAD
			106+*		FOUNDATION EXTENSION
			108+	DC	BL1'00000000' BFTEK,BFALN,HIARCHY
14C			109+	DC	AL3(ERR1) EODAD

ECT CODE	ADDR1	ADDR2	STMT	SOURCE	STATEMENT
00			110+	DC	BL1'00000000' RECFM
			111+	DC	AL3(0) EXLST
			113**		FOUNDATION BLOCK
D4C1C3404040			115+	DC	.CL8'DDMAC' DDNAME
			116+	DC	BL1'00000010' OFLGS
			117+	DC	BL1'00000000' IFLG
			118+	DC	BL2'Q010010000000000' MACR
			120**		BSAM-BPAM-QSAM INTERFACE
01			122+	DC	BL1'00000000' RER1
0001			123+	DC	AL3(1) CHECK, GERR, PERR
			124+	DC	A(1) SYNAD
			125+	DC	H'0' CIND1, CIND2
			126+	DC	AL2(0) BLKSIZE
0000			127+	DC	F'0' WCPO, WCPL, OFFSR, OFFSW
.0001			128+	DC	A(1) IOBA
			129+	DC	AL1(0) NCP
01			130+	DC	AL3(1) EOBR, EOBAD
			132**		BSAM-BPAM INTERFACE
0001			134+	DC	A(1) EOBR
			135+	DC	H'0' DIRECT
			136+	DC	AL2(0) LRECL
0001			137+	DC	A(1) CNTRL, NOTE, POINT
0001	00001		138	ERR1	LA 15,1
A04C	0005E		139	BC	15,RESULT
			140	DS	OD
404040404040			141	NAME	DC CL8' '
D5C4			142	MEND	DC C'MEND'
			143	SAVEAREA	DS 18F
			144	END	
0001			145		=F'1'

ECT CODE	ADDR1	ADDR2	STMT	SOURCE STATEMENT
			1	CREATE CSECT
			2	USING *,15
0 F00C		0000C	3	BC 15,SAVE
			4	DC AL1(7)
9C5C1E3C540			5	DC C'CREATE '
C D00C		0000C	6	SAVE STM 14,12,12(13) SAVE REGISTERS
			7	DROP 15
0			8	BALR 10,0
			9	USING *,10 PROVIDE ADDRESSABIL
0 A1BE		001D0	10	ST 13,SAVEAREA+4 ,SET SAVE AREAS CONV
0			11	LR 11,13
0 A1BA		001CC	12	LA 13,SAVEAREA
B 0008		00008	13	ST 13,8(11)
1			14	LR 9,1 PARAMETER LIST POIN
0 A024		00036	15	PASS1 BC 0,INSTR OPEN DATA SET IN TH
			16	OPEN (MACDCB,(OUTPUT))
0			17+	CNOP 0,4
0 A01E		00030	18+	BAL 1,*,8 LOAD REG1 W/LIST ADDR.
			19+	DC AL1(143) OPTION BYTE
15C			20+	DC AL3(MACDCB) DCB ADDRESS
3			21+	SVC 19 ISSUE OPEN SVC
0 A011		00023	22	MVI PASS1+1,X'F0'
9 000C		0000C	23	INSTR L 3,12(9) LOAD END INDICATO
3 0000		00000	24	LH 3,0(3) LOAD END INDICATOR
3			25	LTR 3,3 SET CONDITION CODE
0 A13A		0014C	26	CLOSE BRANCH IF NON ZERO
9 0004		00004	27	L 3,4(9) LOAD MACRO'S NAME
3 0000		00000	28	L 3,0(3) LOAD MACRO'S NAME A
7 A1AA A202 001BC 00214			29	MVC DIRLIST,BLANKS INITIALISE DIRLIST
5 A1AA 3000 001BC 00000			30	MVC DIRLIST(6),0(3) MOVE MACRO NAME TO
9 0000		00000	31	L 3,0(9) LOAD LIBRARY1 DOPVE
3 0000		00000	32	L 6,0(3) LOAD LIBRARY1 VIRTU
3 0004		00004	33	L 5,4(3) LOAD LIBRARY1 ADDRE
9 0008		00008	34	L 7,8(9) LOAD PTLIB ADDRESS
7 0000		00000	35	L 7,0(7) PTLIB IN 7
0 0001		00001	36	LA 2,1 RECORD COUNTER IN 2
2			37	MR 4,2 I*MULTIPLIER IN 5
5			38	AR 6,5 FIRST RECORD ADDRES
			39	WRITE DECB1,SF,MACDCB,(6)
			40+	CNOP 0,4
0 A07A		0008C	41+	BAL 1,*,24 LOAD DECB ADDRESS
00000			42+	DECB1 DC F'0' EVENT CONTROL BLOCK
			43+	DC X'00' TYPE FIELD
			44+	DC X'20' TYPE FIELD
0			45+	DC AL2(0) LENGTH
0015C			46+	DC A(MACDCB) DCB ADDRESS
00000			47+	DC A(0) AREA ADDRESS
00000			48+	DC A(0) RECORD POINTER WORD
1 000C		0000C	49+	ST 6,12(1,0) STORE AREA ADDRESS
1 0008		00008	50+	L 15,8(1,0) LOAD DCB ADDRESS
0 F030		00030	51+	L 15,48(0,15) LOAD ROWR ROUTINE ADDR
F			52+	BALR 14,15 LINK TO ROWR ROUTINE
			53	* WRITE AND NOTE THE FIRST RECORD
			54	CHECK DECB1
0 A066		00078	55+	LA 1,DECB1 LOAD PARAMETER REG 1

CT CODE	ADDR1	ADDR2	STMT	SOURCE STATEMENT
1008		00008	56+	L 14,8(0,1) PICK UP DCB ADDRESS
E034		00034	57+	L 15,52(0,14) LOAD CHECK ROUT. ADDR.
			58+	BALR 14,15 LINK TO CHECK ROUTINE
			59	NOTE MACDCB
A14A		0015C	60+	LA 1,MACDCB LOAD PARAMETER REG 1
1054		00054	61+	L 15,84(0,1) LOAD NOTE RTN ADDRESS
			62+	BALR 14,15 LINK TO NOTE ROUTINE
A20A		0021C	63	ST 1,WORK1 FIRST RECORD
A1B2	A20A 001C4	0021C	64	MVC DIRLIST+8(3),WORK1 TRACK ADDRESS IN:
A20E		00220	65 LOOP	A 2,=F'1' INCREASE RECORD
			66	CR 2,7 COMPARE RECORD COUN
A0EE		00100	67	BC 10,INSERT BRANCH ON HIGHER OF
			68	AR 6,5 NEXT RECORD ADDRESS
			69 *	WRITE THE CURRENT RECORD
			70	WRITE DECB2,SF,MACDCB,(6)
			71+	CNOP 0,4
ACCE		000E0	72+	BAL 1,*,+24 LOAD DECB ADDRESS
0000			73+DECB2	DC F'0' EVENT CONTROL BLOCK
			74+	DC X'00' TYPE FIELD
			75+	DC X'20' TYPE FIELD
			76+	DC AL2(0) LENGTH
015C			77+	DC A(MACDCB) DCB ADDRESS
0000			78+	DC A(0) AREA ADDRESS
0000			79+	DC A(0) RECORD POINTER WORD
000C		0000C	80+	ST 6,12(1,0) STORE AREA ADDRESS
0008		00008	81+	L 15,8(1,0) LOAD DCB ADDRESS
F030		00030	82+	L 15,48(0,15) LOAD RDWR ROUTINE ADDR
			83+	BALR 14,15 LINK TO RDWR ROUTINE
			84	CHECK DECB2
A0BA		000CC	85+	LA 1,DECB2 LOAD PARAMETER REG 1
1008		00008	86+	L 14,8(0,1) PICK UP DCB ADDRESS
E034		00034	87+	L 15,52(0,14) LOAD CHECK ROUT. ADDR.
			88+	BALR 14,15 LINK TO CHECK ROUTINE
A0AA		000BC	89	BC 15,LOOP RETURN FOR NEXT WRI
A20E		00220	90 INSERT	S 7,=F'1' TOTAL NUMBER OF REC
A20A		0021C	91	ST 7,WORK1 MOVE TOTAL NUMBER OF
A1B6	A20A 001C8	0021C	92	MVC NUMBER,WORK1 RECORDS TO USER'S F
0014		00014	93	L 3,20(9)
0000		00000	94	L 3,0(3)
A20E		00220	95	S 3,=F'1'
A11C		0012E	96	BZ RESULT
			97 *	REPLACE MACRO NAME IN THE DIRECTORY
			98	STOW MACDCB,DIRLIST,R
A14A		0015C	99+	LA 1,MACDCB LOAD PARAMETER REG 1
A1AA		0018C	100+	LA 0,DIRLIST LOAD PARAMETER REG 0
			101+	LCR 1,1 INDICATE TYPE R
			102+	SVC 21 ISSUE STOW SVC
A126		00138	103	BC 15,RECODE
			104 *	ADD MACRO NAME TO THE DATA SET DIRECTORY
			105 RESULT	STOW MACDCB,DIRLIST
A14A		0015C	106+RESULT	LA 1,MACDCB LOAD PARAMETER REG 1
A1AA		0018C	107+	LA 0,DIRLIST LOAD PARAMETER REG 0
			108+	SVC 21 ISSUE STOW SVC
0010		00010	109 RECODE	L 3,16(9)
0000		00000	110	STH 15,0(3) STORE STOW RETURN CODE

ECT CODE	ADDR1	ADDR2	STMT	SOURCE	STATEMENT	
) A18E : D00C :	001D0 0000C	111	RETURN	L	13,SAVEAREA+4	
		112		LM	14,12,12(13)	RESTORE REGISTER
		113		BR	14	
		114	CLOSE	CLOSE	(MACDCB)	
) A142	00154	115+		CNOP	0,4	
		116+	CLOSE	BAL	1,**8 BRANCH AROUND LIST	
		117+		DC	AL1(128) OPTION BYTE	
		118+		DC	AL3(MACDCB) DCB ADDRESS	
) A12E	00140	119+		SVC	20 ISSUE CLOSE SVC	
		120	* THE DATA SET	IS	CLOSED AFTER THE LAST ENTRY IS WR	
		121		BC	15,RETURN	
		122	MACDCB	DCB	DSORG=PO,DDNAME=DDMAC,EODAD=ERR,MACI	
		124+*			DATA CONTROL BLOCK	
		125+*				
		126+		ORG	*-0 TO ELIMINATE UNUSED SPACE	
		127+MACDCB		DS	OF ORIGIN ON WORD BOUNDRY	
		128+		ORG	*+0 TO ORIGIN GENERATION	
		130+*			DIRECT ACCESS DEVICE INTERF	
)00000000000000)0000		132+		DC	BL16'0' FDAD,DVTBL	
		133+		DC	A(0) KEYLE,DEVT,TRBAL	
		135+*			COMMON ACCESS METHOD INTERF	
)01)))0001		137+		DC	AL1(0) BUFNO	
		138+		DC	AL3(1) BUFCB	
		139+		DC	AL2(0) BUFL	
		140+		DC	BL2'00000001000000000' DSORG	
		141+		DC	A(1) IOBAD	
		143+*			FOUNDATION EXTENSION	
)B4)00		145+		DC	BL1'00000000' BFTEK,BFALN,HIARCHY	
		146+		DC	AL3(ERR) EODAD	
		147+		DC	BL1'00000000' RECFM	
		148+		DC	AL3(0) EXLST	
		150+*			FOUNDATION BLOCK	
%D4C1C3404040		152+		DC	CL8'DDMAC' DDNAME	
		153+		DC	BL1'000000010' OFLGS	
		154+		DC	BL1'00000000' IFLG	
		155+		DC	BL2'0000000000100100' MACR	
		157+*			BSAM-BPAM-QSAM INTERFACE	
)01)0001)))0000		159+		DC	BL1'00000000' RER1	
		160+		DC	AL3(1) CHECK, GERR, PERR	
		161+		DC	A(1) SYNAD	
		162+		DC	H'0' CIND1, CIND2	
		163+		DC	AL2(0) BLKSIZE	
		164+		DC	F'0' WCPO, WCPL, OFFSR, OFFSW	

```

GO TO SEARL4
SEARL20
  IF FIND =1 THEN DO
    INTARIT=VALOAR(PID)
    LOCAL=1
    GO TO SEAR2      END
    IF J=1 THEN GOTO SEARWARN1
    ELSE GOTO SEARERR1
SEARL60
  IF FIND =1 THEN DO
    INTARIT=VAGOAR(PID)
    LOCAL=2
SEAR20
  TYPE=0
  RETURN          END
  IF J=2 THEN GOTO D103
  ELSE GOTO SEARERR1
SEARL40
  IF PTLOCHAR1=PTLOCHAR2 THEN GOTO SEAR1
  PID=PTLOCHAR1
SEARL90 IF IDLOCHAR(PID)=INTERNAL THEN GOTO SEARL8
  PID=PID+1
  IF PID)PTLOCHAR2 THEN GOTO SEARL9
SEAR10
  IF PTGBCH=1 THEN GOTO SEARL5
  PID=1
SEARL100
  IF IDGCHAR(PID)=INTERNAL THEN GOTO SEARL11
  PID=PID+1
  IF PID)PTGBCH THEN GOTO SEARL10
SEARL50
  IF FIND=1 THEN DO
    FIND=0
    RETURN      END
    PUT SKIP EDIT(PRINT19,INTERNAL,PRINT1,PTLIB-ADDR,PRINT7,NOM,ERR
OR42,PRINT20)
    (A,X(2),A,X(2),A,F(4),X(2),A,X(2),A,X(2),A)
    GOTO COMMONER
SEARL80
  IF FIND =1 THEN DO
    INTCHAR=VALOCHAR(PID)
    LOCAL=3
    GO TO SEAR3      END
    IF J=3 THEN GOTO SEARWARN1
    ELSE GOTO SEARERR1
SEARL110
  IF FIND =1 THEN DO
    INTCHAR=VAGCHAR(PID)
    LOCAL=4
SEAR30
  TYPE=1
  RETURN      END
  IF J=4 THEN GOTO D103
  ELSE GO TO SEARERR1
SEARWARN10
  PUT SKIP EDIT(PRINT21,PRINT19,INTERNAL,PRINT1,PTLIB-ADDR,PRINT7
,NOM,WARN4)
  (A,X(2),A,X(2),A,X(2),A,F(4),X(2),A,X(2),A,X(2),A)
  FIND=0
  GO TO D103
SEARERR10
  PUT SKIP EDIT(PRINT22,INTERNAL,PRINT1,PTLIB-ADDR,PRINT7,NOM,PRI
NT20)
  (A,X(2),A,X(2),A,F(4),X(2),A,X(2),A,X(2),A)
  GOTO COMMONER
  END
DERR10 PUT SKIP EDIT (ERROR43) (A)
  CM10 PUT EDIT(PRINT1,PTLIB-ADDR,PRINT7,NOM,WARN4) (R(F1))
F10 FORMAT (X(2),A,F(5),X(2),A,X(2),A,X(2),A)
  GO TO COMMONER
DERR20
  PUT SKIP EDIT(ERROR44){A} GO TO CM1
DERR30
  PUT SKIP EDIT(ERROR45) (A) GO TO CM1

```

BIBLIOGRAPHIE

1. Design of Real-Time Computer Systems
James MARTIN
Prentice Hall
2. Programming Languages. Information Structures and Machine Organization
Peter WEGNER
M C Graw Hill
3. Programming Languages : History and Fundamentals
Jean E. SAMMET
Prentice Hall
4. IBM System/360 Operating System
PL/I (F) Programmer's Guide
5. IBM System/360 Operating System
Supervisor and Data Management Services
6. A General Purpose Macro Generator
G. STRACHEY
Computer Journal
7. Computer Software
I. FLORES
Prentice Hall
8. Machines de Traitement de l'Information.
Circuits et Programmes
P. DEBRAINE
Masson

